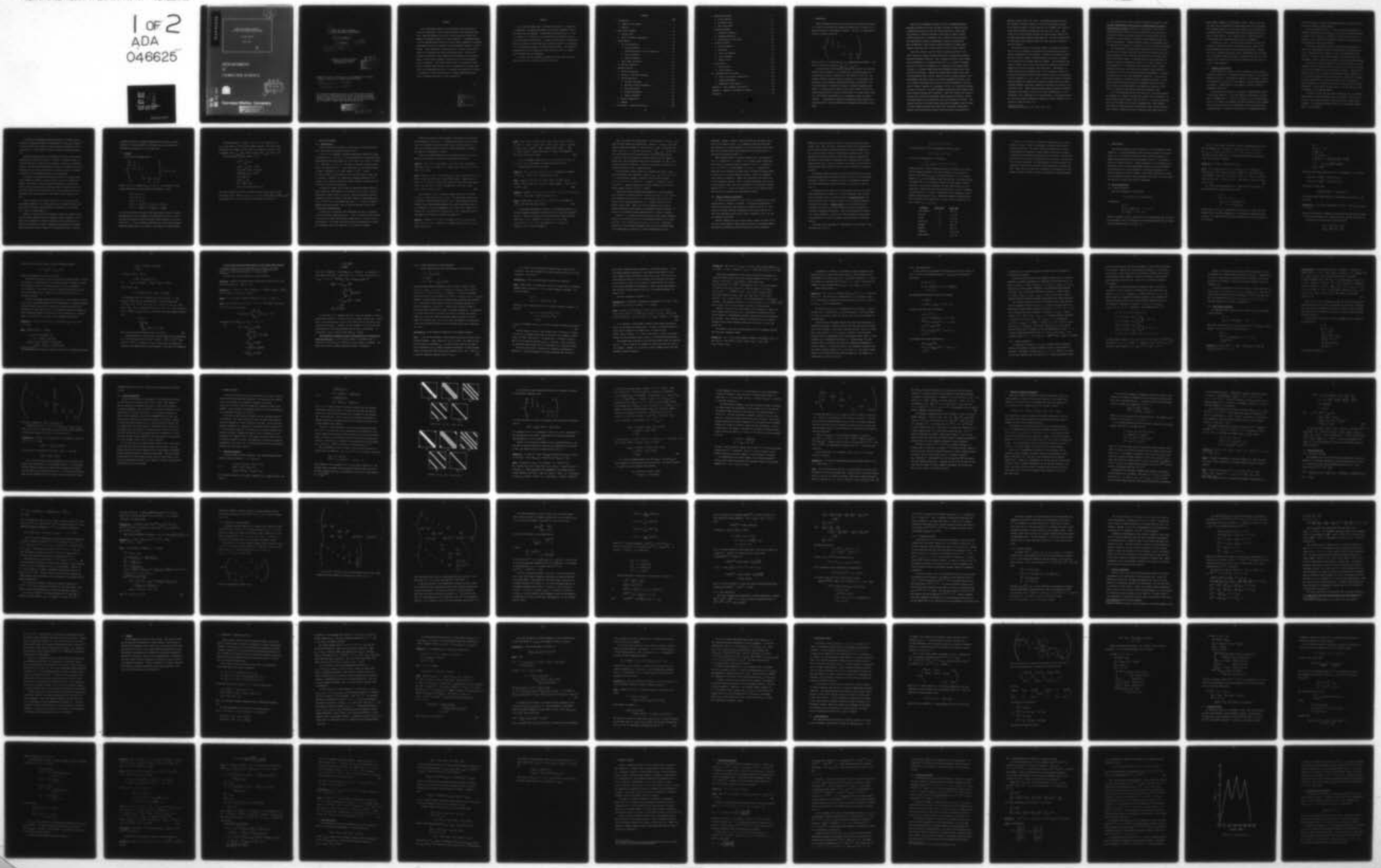
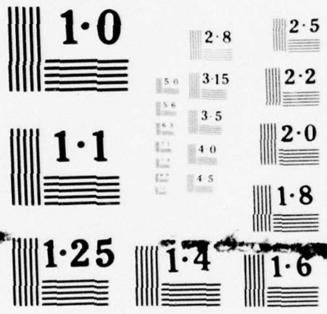
AD-A046 625   CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 12/1
DIRECT AND ITERATIVE METHODS FOR BLOCK TRIDIAGONAL LINEAR SYSTE--ETC(U)
APR 77 D E HELLER   N00014-76-C-0370
UNCLASSIFIED   NL

1 OF 2
ADA
046625




1·0
2·8
2·5
3·15
2·2
3·5
2·0
1·1
4·0
4·5
1·8
1·25
1·4
1·6

DIRECT AND ITERATIVE METHODS FOR BLOCK TRIDIAGONAL LINEAR SYSTEMS

Don Eric Heller

April 1977

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

(6)

DIRECT AND ITERATIVE METHODS FOR BLOCK TRIDIAGONAL LINEAR SYSTEMS.

(10) Don Eric Heller

(11) April 1977 (12) 170p.

(9) Doctoral thesis,

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Carnegie-Mellon University.

(15) N00014-76-C-0370,
✓NSF-MCS-75-22255

This research was supported in part by the Office of Naval Research under Contract N00014-76-C-0370, NR 044-422, by the National Science Foundation under Grant MCS75-222-55, and by the National Aeronautics and Space Administration under Grant NGR-47-102-001, while the author was in residence at ICASE, NASA Langley Research Center.

403 081 LB

ABSTRACT

Block tridiagonal systems of linear equations occur frequently in scientific computations, often forming the core of more complicated problems. Numerical methods for solution of such systems are studied with emphasis on efficient methods for a vector computer. A convergence theory for direct methods under conditions of block diagonal dominance is developed, demonstrating stability, convergence and approximation properties of direct methods. Block elimination (LU factorization) is linear, cyclic odd-even reduction is quadratic, and higher-order methods exist. The odd-even methods are variations of the quadratic Newton iteration for the inverse matrix, and are the only quadratic methods within a certain reasonable class of algorithms. Semi-direct methods based on the quadratic convergence of odd-even reduction prove useful in combination with linear iterations for an approximate solution. An execution time analysis for a pipeline computer is given, with attention to storage requirements and the effect of machine constraints on vector operations.

PREFACE

It is with many thanks that I acknowledge Professor J. F. Traub, for my introduction to numerical mathematics and parallel computation, and for his support and encouragement. Enlightening conversations with G. J. Fix, S. H. Fuller, H. T. Kung and D. K. Stevenson have contributed to some of the ideas presented in the thesis. A number of early results were obtained while in residence at ICASE, NASA Langley Research Center; J. M. Ortega and R. G. Voigt were most helpful in clarifying these results. The thesis was completed while at the Pennsylvania State University, and P. C. Fischer must be thanked for his forbearance.

Of course, the most important contribution has been that of my wife, Molly, who has put up with more than her fair share.

CONTENTS

1. INTRODUCTION

Block tridiagonal matrices are a special class of matrices which arise in a variety of scientific and engineering computations, typically in the numerical solution of differential equations. For now it is sufficient to say that the matrix looks like

$$A = \begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{N-1} & b_{N-1} & c_{N-1} \\ & & & & a_N & b_N \end{pmatrix},$$

where b_i is an $n_i \times n_i$ matrix and a_i, c_i are dimensioned conformally. Thus the full matrix A has dimension $(\sum_{j=1}^{N} n_j) \times (\sum_{j=1}^{N} n_j)$.

In this thesis we study numerical methods for solution of a block tridiagonal linear system $Ax = v$, with emphasis on efficient methods for a vector-oriented parallel computer (e.g., CDC STAR-100, Illiac IV). Our analysis primarily concerns numerical properties of the algorithms, with discussion of their inherent parallelism and areas of application. For this reason many of our results also apply to standard sequential algorithms. A unifying analytic theme is that a wide variety of direct and iterative methods can be viewed as special cases of a general matrix iteration, and we make considerable use of a convergence theory for direct methods. Algorithms are compared using execution time estimates for a simplified model of the CDC STAR-100 and recommendations are made on this basis.

There are two important subclasses of block tridiagonal matrices, depending on whether the blocks are small and dense or large and sparse. A computational method to solve the linear system $Ax = v$ should take into account the internal structure of the blocks in order to obtain storage economy and a low operation count. Moreover, there are important applications in which an approximate solution is satisfactory. Such considerations often motivate iterative methods for large sparse systems, especially when a direct method would be far more expensive.

Since 1965, however, there has been an increased interest in the direct solution of special systems derived from two-dimensional elliptic partial differential equations. With a standard five-point finite difference approximation on a rectangular grid, the a_i, c_i blocks are diagonal and the b_i blocks are tridiagonal, so A possesses a very regular sparse structure. By further specializing the differential equation more structure will be available. Indeed, the seminal paper for many of the important developments dealt with the simplest possible elliptic equation. This was Hockney's work [H5], based on an original suggestion by Golub, on the use of Fourier transforms and the cyclic reduction algorithm for the solution of Poisson's equation on a rectangle. For an $n \times n$ square grid Hockney was able to solve $Ax = v$ in $O(n^3)$ arithmetic operations, while ordinary band-matrix methods required $O(n^4)$ operations. Other $O(n^3)$ methods for Poisson's equation then extant had considerably larger asymptotic constants, and the new method proved to be significantly less time-consuming in practice. Subsequent discovery of the Fast Fourier Transform and Buneman's stable version of cyclic reduction created fast ($O(n^2 \log n)$ operations) and accurate methods that attracted much attention from applications programmers and numerical analysts [B20], [D2], [H6]. The Buneman algorithm has since been extended to Poisson's equation on certain nonrectangular regions* and to general separable elliptic equations on a rectangle [S6], and well-tested Fortran subroutine packages are available (e.g., [S8]). Other recent work on these problems includes the Fourier-Toeplitz methods [F2], and Bank's generalized marching algorithms [B5]. The latter methods work by controlling a fast unstable method, and great care must be taken to maintain stability.

Despite the success of fast direct methods for specialized problems there is still no completely satisfactory direct method that applies to a general nonseparable elliptic equation. The best direct method available for this problem is George's nested dissection [G1], which is theoretically attractive but apparently hard to implement. Some interesting techniques to improve this situation are discussed by Eisenstat, Schultz and Sherman [E1] and Rose and Whitten [R5] among others. Nevertheless, in present practice the nonseparable case is still frequently solved by an iteration, and excellent methods based on direct solution of the separable case have appeared (e.g., [C4], [C5]). The multi-level adaptive techniques recently analyzed and considerably developed by Brandt [B13] are a rather different approach, successfully combining iteration and a sequence of discretization grids. Brandt discusses several methods suitable for parallel computation and although his work has not been considered here, it will undoubtedly play an important role in future studies. The state-of-the-art for solution of partial differential equations on vector computers is summarized by Ortega and Voigt [O1].

*References include [B19], [D1], [S5], [S7], [S9].

With the application area of partial differential equations in mind, the major research goal of this thesis is a general analysis of direct block elimination methods, regardless of the sparsity of the blocks. As a practical measure, however, the direct methods are best restricted to cases where the blocks are small enough to be stored explicitly as full matrices, or special enough to be handled by some storage-efficient implicit representation. Proposed iterative methods for nonseparable elliptic equations will be based on the ability to solve such systems efficiently.

Our investigation of direct methods includes the block LU factorization, the cyclic reduction algorithm (called odd-even reduction here) and Sweet's generalized cyclic reduction algorithms [S10], along with variations on these basic themes. The methods are most easily described as a sequence of matrix transformations applied to A in order to reduce it to block diagonal form. We show that, under conditions of block diagonal dominance, matrix norms describing the off-diagonal blocks relative to the diagonal blocks will not increase, decrease quadratically, or decrease at higher rates, for each basic algorithm, respectively. Thus the algorithms are naturally classified by their convergence properties. For the cyclic reduction algorithms, quadratic convergence leads to the definition of useful semi-direct methods to approximate the solution of $Ax = v$. Some of our results in the area have previously appeared in [H2].

All of the algorithms discussed here can be run, with varying degrees of efficiency, on a parallel computer that supports vector operations. By this we mean an array processor, such as Illiac IV, or a pipeline processor, such as the CDC STAR-100 or the Texas Instruments ASC. These machines fall within the single instruction stream-multiple data stream classification of parallel computers, and have sufficiently many common features to make a general comparison of algorithms possible. However, there are also many special restrictions or capabilities that must be considered when designing programs for a particular machine. Our comparisons for a simplified model of STAR indicate how similar comparisons should be made for other machines. For a general survey of parallel methods in linear algebra plus background material, [H3] is recommended.

In almost every case, a solution method tailored to a particular problem on a particular computer can be expected to be superior to a general method naively applied to that problem and machine. However, the costs of tailoring can be quite high. Observations about the behavior of the general method can be used to guide the tailoring for a later solution, and we believe that the results presented here will allow us to move in that direction.

1.A. Summary of Main Results

Direct methods for solution of block tridiagonal linear systems are discussed in Sections 3-6, semidirect methods in Section 7, and iterative methods in Section 8. Preliminaries (analytic tools and models of parallel computation) are in Section 2 and remarks on certain applications are in Section 9. Implementation of algorithms on a parallel computer is discussed in Section 10.

We consider direct methods as a sequence of transformations applied to the linear system $Ax = v$ with the intention of simplifying its structure. The matrix $B[A] = I - (D[A])^{-1}A$, where $D[A]$ is the block diagonal part of A, is shown to be important for stability analysis, error propagation in back substitutions, and approximation errors in semidirect and iterative methods. We appear to be the first to systematically exploit

$B[A]$ in the analysis of direct and semidirect methods, though it has long been used to analyze iterative methods.

Under the assumption $\| B[A] \| < 1$, when a direct method generates a sequence of matrices M_i with $\| B[M_{i+1}] \| \le \| B[M_i] \|$, we classify the method as linear, and if $\| B[M_{i+1}] \| \le \| B[M_i]^2 \|$ or $\| B[M_i] \|^2$ we say that it is quadratic. The ∞-norm is used throughout; extension to other norms would be desirable but proves quite difficult.

The block LU factorization and block Gauss-Jordan elimination are shown to be linear methods, and their properties are investigated in Section 3. Sample results are stability of the block factorization when $\| B[A] \| < 1$, bounds on the condition numbers of the pivotal blocks, and convergence to zero of portions of the matrix in the Gauss-Jordan algorithm.

The methods of odd-even elimination and odd-even reduction, which are inherently parallel algorithms closely related to the fast Poisson solvers, are shown to be quadratic, and are discussed in Section 4. Odd-even elimination is seen to be a variation of the quadratic Newton iteration for A^{-1}, and odd-even reduction can be considered as a special case of Hageman and Varga's general cyclic reduction technique [H1], as equivalent to block elimination on a permuted system, and as a compact form of odd-even elimination. We also discuss certain aspects of the fast Poisson solvers as variations of the odd-even methods.

In Section 5 we show that the odd-even methods are the only quadratic methods within a certain reasonable class of algorithms. This class includes all the previous methods as particular cases of a general matrix iteration. The relationship between block fill-in and quadratic convergence is also discussed.

In Section 6 the quadratic methods are shown to be special cases of families of higher-order methods, characterized by $\| B[M_{i+1}] \| \le \| B[M_i] \|^p$. We note that the p-fold reduction (Sweet's generalized reduction [S10]) is equivalent to a 2-fold reduction (odd-even) using a coarser partitioning of A.

Semidirect methods based on the quadratic properties of the odd-even algorithms are discussed in Section 7. Briefly, a semidirect method is a direct method that is stopped before completion, producing an approximate solution. We analyze the approximation error and its propagation vis-a-vis rounding error in the back substitution of odd-even reduction. Practical limitations of the semidirect methods are also considered.

In Section 8 we consider some iterative methods for systems arising from differential equations, emphasizing their utility for parallel computers. Fast Poisson solvers and general algorithms for systems with small blocks form the basis for iterations using the natural, multi-line and spiral orderings of a rectangular grid. We also remark on the use of elimination and the applicability of the Parallel Gauss iteration [T2], [H4].

A brief section on applications deals with two situations in which the assumption $\|B[A]\| < 1$ is not met, namely curve fitting and finite elements, and in the latter case we describe a modification to the system $Ax = v$ that will obtain $\|B[A]\| < 1$.

Methods for solution of block tridiagonal systems are compared in Section 10 using a simplified model of a pipeline (vector) computer. We conclude that a combination of odd-even reduction and LU factorization is a powerful direct method, and that a combination of semidirect and iterative methods may provide limited savings if an approximate solution is desired.

A general discussion of storage requirements and the effect of machine constraints on vector operations is included; these are essential considerations for practical use of the algorithms.

1.B. Notation

Let the block tridiagonal matrix

$$A = \begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{N-1} & b_{N-1} & c_{N-1} \\ & & & & a_N & b_N \end{pmatrix} = (a_j, b_j, c_j)_N.$$

Here b_i is an $n_i \times n_i$ matrix and $a_1 = 0$, $c_N = 0$. The subscript N in the triplex notation will often be omitted for simplicity. Define

$$\begin{aligned} L[A] &= (a_j, 0, 0), \\ D[A] &= (0, b_j, 0), \\ U[A] &= (0, 0, c_j), \\ B[A] &= I - (D[A])^{-1}A = (-b_j^{-1} a_j, 0, -b_j^{-1} c_j), \\ C[A] &= I - A(D[A])^{-1} = (-a_j b_{j-1}^{-1}, 0, -c_j b_{j+1}^{-1}). \end{aligned}$$

Use of these notations for general partitioned matrices should be evident; nonsingularity of D[A] is assumed in the definitions of B and C. B[A] is the matrix associated with the block Jacobi linear iteration for the solution of $Ax = v$. It also assumes an important role in the study of certain semidirect methods and will be central to our analysis of direct methods.

We further define $E_j = \mathrm{diag}(0,\dots,0, I, 0,\dots,0)$, where the I is $n_j \times n_j$ and occurs in the jth block diagonal position. Pre- (post-) multiplication by E_j selects the jth block row (column) of a matrix, so that, for example, $D[A] = \sum_{i=1}^{N} E_i A E_i$, $D[A]E_j = E_j D[A]$, and $\sum_{j=i}^{N} E_i\, L[A]\, E_j = 0$.

For an n×n matrix M and an n-vector v, let

$$\rho_i(M) = \sum_{j=1}^{n} |m_{ij}|,$$

$$\gamma_j(M) = \sum_{i=1}^{n} |m_{ij}| = \rho_j(M^T),$$

$$\|M\| = \max_i \rho_i(M), \text{ the } \infty\text{-norm},$$

$$\|M\|_1 = \max_j \gamma_j(M) = \|M^T\|,$$

$$\|v\| = \max_i |v_i|,$$

$$\|v\|_1 = \sum_{i=1}^{n} |v_i|,$$

$$|M| = (|m_{ij}|), \text{ and}$$

$$\rho(M) = \text{the spectral radius of } M.$$

The reader should take care not to confuse row sums ($\rho_i(M)$) with the spectral radius ($\rho(M)$). When $A = (a_j, b_j, c_j)$, let $\lambda(A) = \max_j (4\|b_j^{-1} a_j\|\,\|b_{j-1}^{-1} c_{j-1}\|)$. The condition $\lambda(A) \le 1$ will be used as an alternative to $\|B[A]\| < 1$.

2. SOME INITIAL REMARKS

2.A. Analytic Tools

Much of the analysis presented in later sections involves the matrix $B[A]$. Why should we consider this matrix at all?

First, it is a reasonably invariant quantity which measures the diagonal dominance of A. That is, if E is any nonsingular block diagonal matrix conforming with the partitioning of A, let $A^* = EA$, so $B^* = B[A^*] = I - D^{*-1}A^* = I - (ED)^{-1}(EA) = I - D^{-1}A = B$. Thus $B[A]$ is invariant under block scalings of the equations $Ax = v$. On the other hand, B is not invariant under a change of variables $y = Ex$, which induces $A' = AE^{-1}$. We have $B' = B[A'] = EBE^{-1}$, and for general A and E only the spectral radius $\rho(B)$ is invariant. Similarly, $C[A]$ measures the diagonal dominance of A by columns, and is invariant under block diagonal change of variables but not under block scaling of the equations.

Secondly, one simple class of parallel semidirect methods (Section 7) can be created by transforming $Ax = v$ into $A^*x = v^*$, $A^* = MA$, $v^* = Mv$ for some matrix M, where $\|B[A^*]\| \ll \|B[A]\|$. This transformation typically represents the initial stages of some direct method. An approximate solution $y = D[A^*]^{-1}v^*$ is then computed with error $x - y = B[A^*]x$, so $\|B[A^*]\|$ measures the relative error. Thus we study the behavior of $B[A]$ under various matrix transformations.

Thirdly, $B[A]$ is the matrix which determines the rate of convergence of the block Jacobi iteration for $Ax = v$. This is a natural parallel iteration, as are the faster semi-iterative methods based on it when A is positive definite. It is known that various matrix transformations can affect the convergence rates of an iteration, so we study this property.

Finally, $B[A]$ and $C[A]$ arise naturally in the analysis of block elimination methods to solve $Ax = v$. C represents multipliers used in elimination, and B represents similar quantities used in back substitution. In order to place upper bounds on the size of these quantities we consider the effects of elimination on $\|B[A]\|$.

Since we will make frequent use of the condition $\|B[A]\| < 1$ we now discuss some conditions on A which will guarantee that it holds.

Lemma 2.1. Let $J = J_1 + J_2$ be order n, $|J| = |J_1| + |J_2|$, and suppose K satisfies $K = J_1K + J_2$. Then $\|J\| < 1$ implies $\rho_i(K) \le \rho_i(J)$ for $1 \le i \le n$, and so $\|K\| \le \|J\|$.

Proof. From $|J_1| + |J_2| = |J|$ we have $\rho_i(J_1) + \rho_i(J_2) = \rho_i(J)$ for $1 \le i \le n$. From $K = J_1K + J_2$ we have $\rho_i(K) \le \rho_i(J_1)\|K\| + \rho_i(J_2)$. Suppose $\|K\| \ge 1$. Then $\rho_i(K) \le \rho_i(J_1)\|K\| + \rho_i(J_2)\|K\| = \rho_i(J)\|K\| \le \|J\|\ \|K\|$, which implies $\|K\| \le \|J\|\ \|K\|$ and $1 \le \|J\|$, a contradiction. Thus $\|K\| < 1$ and $\rho_i(K) \le \rho_i(J_1) + \rho_i(J_2) = \rho_i(J)$. QED.

We note that if $\rho_i(J_1) \neq 0$ then we actually have $\rho_i(K) < \rho_i(J)$ and if $\rho_i(J_1) = 0$ then $\rho_i(K) = \rho_i(J_2) = \rho_i(J)$. It follows that if each row of J_1 contains a nonzero element then $\|K\| < \|J\|$. These conditions are reminiscent of the Stein-Rosenberg Theorem ([V3], [H8]) which states that if J is irreducible, J_1 and J_2 nonnegative, $J_2 \neq 0$ and $\rho(J_1) < 1$ then $\rho(J) < 1$ implies $\rho(K) < \rho(J)$ and $\rho(J) = 1$ implies $\rho(K) = \rho(J)$. There are also correspondences to the theory of regular splittings [V3].

Lemma 2.2. Suppose $J = (I - S)K + T$ is order n, $|J| = |(I - S)K| + |T|$, $\rho_i(S) \le \rho_i(T)$, $1 \le i \le n$. Then $\|J\| < 1$ implies $\rho_i(K) \le \rho_i(J)$, $1 \le i \le n$, and so $\|K\| \le \|J\|$.

Proof. For $1 \le i \le n$, $\rho_i(T) \le \rho_i(T) + \rho_i((I - S)K) = \rho_i(J) \le \|J\| < 1$, so $\rho_i(T) < 1$. Now, $1 > \rho_i(J) = \rho_i((I - S)K) + \rho_i(T) \ge \rho_i(K) - \rho_i(S)\|K\| + \rho_i(T) \ge \rho_i(K) + \rho_i(T)(1 - \|K\|)$. Suppose $\|K\| = \rho_j(K) > 1$. Then $1 > \rho_j(K) + \rho_j(T)(1 - \rho_j(K))$, and we have $1 < \rho_j(T)$, a contradiction. Thus $\|K\| \le 1$ and $\rho_i(J) \ge \rho_i(K)$. QED.

If $S = T$ in Lemma 2.2 then $K = SK + (J - S)$, $|J| = |(I - S)K| + |S| = |J - S| + |S|$, so by Lemma 2.1 we again conclude $\|K\| \le \|J\|$.

Similar results hold for the 1-norm.

Lemma 2.3. Let $J = J_1 + J_2$, $|J| = |J_1| + |J_2|$, and suppose K satisfies $K = KJ_1 + J_2$. Then $\|J\|_1 < 1$ implies $\|K\|_1 \le \|J\|_1$.

Proof. Since $J^T = J_1^T + J_2^T$, $|J^T| = |J_1^T| + |J_2^T|$, $K^T = J_1^T K^T + J_2^T$ and $\|M\|_1 = \|M^T\|$, we have $\|J^T\| = \|J\|_1 < 1$, and by Lemma 2.1, $\|K\|_1 = \|K^T\| \le \|J^T\| = \|J\|_1$. QED.

Lemma 2.4. Suppose $J = K(I - S) + T$, $|J| = |K(I - S)| + |T|$, $\gamma_i(S) \le \gamma_i(T)$, $1 \le i \le n$. Then $\|J\|_1 < 1$ implies $\|K\|_1 \le \|J\|_1$.

Proof. We have $\gamma_i(M) = \rho_i(M^T)$, $J^T = (I - S^T)K^T + T^T$, so by Lemma 2.2 $\|K^T\| \le \|J^T\|$, and the result follows. QED.

To investigate the condition $\|B[A]\| < 1$, we first note that if A is strictly diagonally dominant by rows then $\|B[A]\| < 1$ under any partitioning of A. Indeed, if we take J to be the point Jacobi linear iteration matrix for A, $J = I - \text{diag}(A)^{-1}A$, $J_1 = D[J]$, $J_2 = J - J_1$, then $\|J_1\| \le \|J\| < 1$, $|J| = |J_1| + |J_2|$ and $B[A] = (I - J_1)^{-1}J_2$, so $\|B[A]\| \le \|J\| < 1$ by use of Lemma 2.1.

Now, if we denote the partitioning of A by the vector $\Pi = (n_1, n_2, \dots, n_N)$ then the above remark may be generalized. Suppose $\Pi' = (m_1, m_2, \dots, m_M)$ represents another partitioning of A which refines Π. That is, there are integers $1 = p_0 < p_1 < \dots < p_N = M+1$ such that $n_i = \sum_{j=p_{i-1}}^{p_i - 1} m_j$. If B[A] and B'[A] are defined by the two partitions and $\| B'[A] \| < 1$ then $\| B[A] \| \le \| B'[A] \|$. This is an instance of the more general rule for iterative methods that "more implicitness means faster convergence" (cf. [H8, p. 107]) and will be useful in Section 6. Briefly, it shows that one simple way to decrease $\| B[A] \|$ is to use a coarser partition.

Let us consider a simple example to illustrate these ideas. Using finite differences, the differential equation $-u_{xx} - u_{yy} + \lambda u = f$ on $[0,1]^2$, $\lambda > 0$, with Dirichlet boundary conditions, gives rise to the matrix $A = (-I, M, -I)$, $M = (-1, 4+\lambda h^2, -1)$. By Lemma 2.1, $\| B[A] \| \le \| J[A] \| = 4/(4+\lambda h^2) < 1$. Actually, since B[A] is easily determined we can give the better estimate $\| B[A] \| = 2 \| M^{-1} \| < 2/(2+\lambda h^2)$.

Indeed, many finite difference approximations to elliptic boundary value problems will produce matrices A such that $\| B[A] \| < 1$ [V3]; $\| B[A] \|$ will frequently be quite close to 1. This condition need not hold for certain finite element approximations, so a different analysis will be needed in those cases. Varah [V1], [V2] has examples of this type of analysis which are closely related to the original continuous problem. In Section 9.B we consider another approach based on a change of variable.

$\| B[A] \| < 1$ is a weaker condition than strict block diagonal dominance relative to the ∞-norm, which for block tridiagonal matrices is the condition ([F1], [V1]) $\| b_j^{-1} \| (\| a_j \| + \| c_j \|) < 1$, $1 \le j \le N$. The general definition of block diagonal dominance allows the use of different norms, so for completeness we should also consider theorems based on this assumption. However, in order to keep the discussion reasonably concise this will not always be done. It is sometimes stated that block diagonal dominance is the right assumption to use when analyzing block elimination, but we have found it more convenient and often more general to deal with the assumption $\|B[A]\| < 1$.

Other concepts related to diagonal dominance are G- and H-matrices. If $J = I - \text{diag}(A)^{-1}A = L + U$, where L (U) is strictly lower (upper) triangular, then A is strictly diagonally dominant if $\|J\| < 1$, a G-matrix if $\|(I - |L|)^{-1}|U|\| < 1$, and an H-matrix if $\rho(|J|) < 1$. Ostrowski [O2] shows that A is an H-matrix iff there exists a nonsingular diagonal matrix E such that AE is strictly diagonally dominant. Varga [V4] summarizes recent results on H-matrices. Robert [R3] generalizes these definitions to block diagonally dominant, block G- and block H-matrices using vectorial and matricial norms, and proves convergence of the classical iterative methods for $Ax = v$. Again, for simplicity we will not be completely general and will only consider some representative results.

2.B. Models of Parallel Computation

In order to read Sections 3-9 only a very loose conception of a vector-oriented parallel computer is necessary. A more detailed description is needed for the execution time comparison of methods in Section 10. This subsection contains sufficient information for the initial part of the text, while additional details may be found in Appendix A, [H3], [S3], and the references given below.

Parallel computers fall into several general classes, of which we consider the vector computers, namely those operating in a synchronous manner and capable of performing efficient floating point vector operations.

Examples are the array processor Illiac IV, with 64 parallel processing elements [B6], [B12]; the Cray-1, with eight 64-word vector registers [C8]; and the pipeline processors CDC STAR-100 [C2] and Texas Instruments ASC [W1], which can handle vectors of (essentially) arbitrary length. These machines have become important tools for large scientific computations, and there is much interest in algorithms able to make good use of special machine capabilities. Details of operation vary widely, but there are sufficiently many common features to make a general comparison of algorithms possible.

An important parallel computer not being considered here is the asynchronous Carnegie-Mellon University C.mmp [W4], with up to 16 minicomputer processors. Baudet [B7] discusses the theory of asynchronous iterations for linear and nonlinear systems of equations, plus results of tests on C.mmp.

Instructions for our model of a vector computer consist of the usual scalar operations and a special set of vector operations, some of which will be described shortly. We shall consider a conceptual vector to be simply an ordered set of storage locations, with no particular regard to the details of storage. A machine vector is a more specialized set of locations valid for use in a vector instruction, usually consisting of locations in arithmetic progression. The CDC STAR restricts the progression to have increment = 1, which often forces programs to perform extra data manipulation in order to organize conceptual vectors into machine vectors.

We denote vector operations by a parenthesized list of indices. Thus the vector sum $w = x+y$ is

$$w_i = x_i + y_i, \ (1 \le i \le N)$$

if the vectors are in R^N, the componentwise product would be

$$w_i = x_i \times y_i, \ (1 \le i \le N),$$

and a matrix multiplication C = AB would be

$$c_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj}, \ (1 \le i \le N; \ 1 \le j \le N).$$

Execution time for a vector operation depends on the length of the vector and the internal mechanism of the vector computer, but can usually be written in the form $\tau_{op}N + \sigma_{op}$ if the vector length is N. On a pipeline computer, $1/\tau_{op}$ is the asymptotic result rate and σ_{op} is the vector instruction startup cost. On a k-parallel computer (one with either k parallel processors or k-word vector registers) the basic time would be t_{op} for k operations in parallel and the vector instruction cost $t_{op} \lceil N/k \rceil$ + overhead. In either case the $\tau N + \sigma$ model is adequate for vector operations; scalar operation times are given as t_{op}. Simplified instruction execution times for the CDC STAR-100 are given below, in terms of the 40 nanosecond cycle time. Usually $\tau_{op} \ll t_{op} \ll \sigma_{op}$.

operation	scalar time	vector time
add, subtract	13	$\frac{1}{2}N + 71$
multiply	17	$N + 159$
divide	46	$2N + 167$
square root	74	$2N + 155$
transmit	11	$\frac{1}{2}N + 91$
maximum	-	$6N + 95$
summation	-	$6.5N + 122$
inner product	-	$6N + 130$

The use of long vectors is essential for effective use of a vector computer, in order to take full advantage of increased result rates in the vector operations. The rather large vector startup costs create a severe penalty for operations on short vectors, and one must be careful to design algorithms which keep startup costs to a minimum. Scalar operations can also be a source of inefficiency even when most of the code consists of vector operations. Data manipulation is often an essential consideration, especially when there are strong restrictions on machine vectors. These issues and others are illustrated in our comparison of algorithms in Section 10, which is based on information for the CDC STAR-100.

3. LINEAR METHODS

The simplest approach to direct solution of a block tridiagonal linear system $Ax = v$ is block elimination, which effectively computes an LU factorization of A. We discuss the process in terms of block elimination on a general partitioned matrix under the assumption $\|B[A]\| < 1$; the process is shown to be linear and stable. Various properties of block elimination are considered, such as pivoting requirements, bounds on the condition numbers of the pivotal blocks, error propagation in the back substitution, and special techniques for special systems. Gauss-Jordan elimination is studied for its numerical properties, which foreshadow our analysis of the more important odd-even elimination and reduction algorithms.

3.A. The LU Factorization

3.A.1. Block Elimination

The block tridiagonal LU factorization

$$A = (a_j, b_j, c_j)_N = (\ell_j, I, 0)_N (0, d_j, c_j)_N$$

is computed by

$$\begin{aligned} d_1 &= b_1 \\ d_j &= b_j - a_j d_{j-1}^{-1} c_{j-1}, \quad j = 2, \ldots, N, \\ \ell_j &= a_j d_{j-1}^{-1}, \quad j = 2, \ldots, N, \end{aligned}$$

where the existence of d_{j-1}^{-1} is assumed and will be demonstrated when $\|B[A]\| < 1$. For the moment we are not concerned with the particular methods used to compute or avoid computing either ℓ_j or $d_{j-1}^{-1} c_{j-1}$.

Let $A_1 = A$ and consider the matrix A_2 which is obtained after the first block elimination step. We have $A_2 = (\alpha_j, \beta_j, c_j)_N$, $\beta_1 = d_1 = b_1$, $\alpha_2 = 0$, $\beta_2 = d_2$, $\alpha_j = a_j$ and $\beta_j = b_j$ for $3 \le j \le N$. Let $B_1 = B[A_1]$, $B_2 = B[A_2]$. Note that B_1 and B_2 differ only in the second block row; more specifically, in the (2,1) and (2,3) positions.

Theorem 3.1. If $\| B_1 \| < 1$ then $\| B_2 \| \le \| B_1 \|$.

Proof. Let $T = B_1 E_1$ and let $S = TB_1$. We have $d_2 = b_2(I - b_2^{-1} a_2 b_1^{-1} c_1)$, $\| b_2^{-1} a_2 b_1^{-1} c_1 \| = \| S \| \le \| B_1 \| \, \| E_1 \| \, \| B_1 \| = \| B_1 \|^2 < 1$, so d_2^{-1} exists and B_2 is well-defined. Also, $B_1 = (I - S)B_2 + T$, $|B_1| = |(I - S)B_2| + |T|$ by construction, and $\rho_j(S) \le \rho_j(T) \| B_1 \| \le \rho_j(T)$. Thus Lemma 2.2 applies, yielding $\| B_2 \| \le \| B_1 \|$. QED

Now, for any partitioned matrix the N stages of block elimination (no block pivoting) can be described by the process

```
A_1 = A
for i = 1,...,N-1
   | A_{i+1} = (I - L[A_i]E_i D[A_i]^{-1})A_i
   |         = (I + L[C[A_i]]E_i)A_i.
```

We transform A_i into A_{i+1} by eliminating all the blocks of column i below the diagonal; when A is block tridiagonal A_N is the upper block triangular matrix $(0, d_j, c_j)_N$. The same effect can be had by eliminating the blocks individually by the process

```
$A_1 = A$
for $i = 1,\dots,N-1$
    $A_{i+1}^{(i)} = A_i$
    for $j = i+1,\dots,N$
        $A_{i+1}^{(j)} = (I - E_j A_{i+1}^{(j-1)} E_i D[A_{i+1}^{(j-1)}]^{-1}) A_{i+1}^{(j-1)}$
                $= (I + E_j C[A_{i+1}^{(j-1)}] E_i) A_{i+1}^{(j-1)}$
    $A_{i+1} = A_{i+1}^{(N)}$.
```

Each minor stage affects only block row j of the matrix. It is seen that

$$\text{block row } j \text{ of } A_{i+1}^{(j-1)} = \text{block row } j \text{ of } A_i,$$
$$\text{block row } i \text{ of } A_{i+1}^{(j-1)} = \text{block row } i \text{ of } A_i,$$

from which it follows that

$$E_j A_{i+1}^{(j-1)} E_i D[A_{i+1}^{(j-1)}]^{-1} = E_j A_i E_i D[A_i]^{-1},$$

so the major stages indeed generate the same sequence of matrices A_i. Let $B_i = B[A_i]$.

<u>Corollary 3.1.</u> If A is block tridiagonal and $\|B_1\| < 1$ then $\|B_{i+1}\| \le \|B_i\|$, $1 \le i \le N-1$.

The proof of Corollary 3.1 requires a bit more care than the proof of Theorem 3.1 would seem to indicate; this difficulty will be resolved in Theorem 3.2.

We should also note the trivial "error relations"

$$\|A_{i+1} - A_N\| \le \|A_i - A_N\|,$$
$$\|B_{i+1} - B_N\| \le \|B_i - B_N\|.$$

These follow from the fact that, for block tridiagonal matrices,

$$A_i = \sum_{j=1}^{i} E_j A_N + \sum_{j=i+1}^{N} E_j A_1,$$

which holds regardless of the value of $\|B_1\|$.

Wilkinson [W3] has given a result for Gaussian elimination that is similar to Corollary 3.1, namely that if A is strictly diagonally dominant by columns then the same is true for all reduced matrices and $\|A_{i+1}\text{ (reduced)}\|_1 \le \|A_i\text{ (reduced)}\|_1$. This actually applies to arbitrary matrices and not just the block tridiagonal case. Brenner [B14] anticipated the result but in a different context.

Our next two theorems, generalizing Corollary 3.1 and Wilkinson's observation, lead to the characterization of block elimination as a norm-non-increasing projection method*, and proves by bounding the growth factors that block elimination is stable when $\|B[A]\| < 1$. In addition, many special schemes for block tridiagonal matrices are equivalent to block elimination on a permuted matrix, so the result will be useful in some immediate applications.

<u>Theorem 3.2.</u> If $\|B_1\| < 1$ then $\|B_{i+1}\| \le \|B_i\|$ and $\|A_{i+1}\| \le \|A_i\|$ for $1 \le i \le N-1$.

<u>Proof</u>. Suppose $\|B_i\| < 1$. We have

$$\begin{aligned} A_{i+1} &= (I - L[A_i]E_i D[A_i]^{-1})A_i \\ &= A_i - L[A_i]E_i + L[A_i]E_i B_i, \\ \rho_j(A_{i+1}) &\le \rho_j(A_i - L[A_i]E_i) + \rho_j(L[A_i]E_i B_i) \\ &\le \rho_j(A_i - L[A_i]E_i) + \rho_j(L[A_i]E_i)\|B_i\| \end{aligned}$$

*The projections are onto spaces of matrices with zeros in appropriate positions.

$$\begin{aligned} &\le \rho_j(A_i - L[A_i]E_i) + \rho_j(L[A_i]E_i) \\ &= \rho_j(A_i), \end{aligned}$$

so $\| A_{i+1} \| \le \| A_i \|$. Now let

$$Q = D[A_i]^{-1} L[A_i]E_i D[A_i]^{-1} A_i,$$

so $\qquad A_{i+1} = A_i - D[A_i]Q, \qquad D[A_{i+1}] = D[A_i](I - D[Q]).$

Let $S = D[Q]$. Since

$$Q = (-L[B_i]E_i)(I - B_i) = -L[B_i] + L[B_i]E_iB_i,$$

$S = D[L[B_i]E_iB_i]$ and $\| S\| \le \| L[B_i]\| \, \| E_i\| \, \| B_i\| \le \| B_i\|^2 < 1$. Thus $D[A_{i+1}]^{-1}$ exists and B_{i+1} is well-defined. It may be verified that $(I - S)B_{i+1} = B_i - S + Q$. Let $T = B_i - S + Q$, $J = S + T = B_i + Q$. Since $D[B_i] = D[Q - S] = 0$, we have $D[T] = 0$ and $|J| = |S| + |T|$. Thus, by Lemma 2.1, $\| J\| < 1$ implies $\| B_{i+1}\| \le \| J\|$. But we have, after some algebraic manipulation,

$$\begin{aligned} J &= B_i + Q \\ &= \sum_{k=1}^{i} E_kB_i \\ &\quad + \sum_{k=i+1}^{N} E_kB_i(I - E_i + E_iB_i), \end{aligned}$$

and it follows from this expression that $\| J\| \le \| B_i\|$. QED

Although expressed in terms of the N-1 major stages of block elimination, it is clear that we also have, for $B_i^{(j)} = B[A_i^{(j)}]$, $\| B_i^{(j)} \| \le \| B_i^{(j-1)}\|$ and $\| A_i^{(j)} \| \le \| A_i^{(j-1)} \|$. Moreover, as long as blocks below the diagonal are eliminated such inequalities will hold regardless of the order of elimination.

We shall adopt the name "linear method" for any process which generates a sequence of matrices M_i such that $\| B[M_{i+1}] \| \le \| B[M_i] \|$, given that $\| B[M_1] \| < 1$. Thus block elimination is a linear method.

A result that is dual to Theorem 3.2 is the following.

Definition. Let $\mathcal{A}_i$ be the reduced matrix consisting of the last N-i+1 block rows and columns of A_i. ($\mathcal{A}_1 = A_1 = A$).

Theorem 3.3. If $\| C[A] \|_1 < 1$ then $\| C[\mathcal{A}_{i+1}] \|_1 \le \| C[\mathcal{A}_i] \|_1$, $\| \mathcal{A}_{i+1} \|_1 \le \| \mathcal{A}_i \|_1$, and $\| L[C_i] E_i \|_1 \le \| C[A] \|_1$, $C_i = C[A_i]$.

Remark. It is not true in general that $\| C_{i+1} \|_1 \le \| C_i \|_1$ if $\| C_1 \|_1 < 1$.

Proof. Let the blocks of C_i be c_{mp}, $1 \le m, p \le N$. Then $C[\mathcal{A}_i] = (c_{mp})_{i \le m, p \le N}$.
Let $K = (c_{mp} + c_{mi} c_{ip})_{i+1 \le m, p \le N}$

$$= (c_{mp})_{i+1 \le m, p \le N} + \begin{pmatrix} c_{i+1,i} \\ \vdots \\ c_{Ni} \end{pmatrix} (c_{i,i+1} \cdots c_{iN}).$$

Using Lemma 2.4, $\| C[\mathcal{A}_{i+1}] \|_1 \le \| K \|_1$ if $\| K \|_1 < 1$.

But
$$\begin{aligned}
\gamma_q(K) &\le \gamma_q\Big(\sum_{k=i+1}^{N} E_k \, C[\mathcal{A}_i]\Big) \\
&\quad + \left\| \begin{pmatrix} c_{i+1,i} \\ \vdots \\ c_{Ni} \end{pmatrix} \right\|_1 \gamma_q(E_i \, C[\mathcal{A}_i]) \\
&\le \gamma_q\Big(\sum_{k=i+1}^{N} E_k \, C[\mathcal{A}_i]\Big) \\
&\quad + \| C[\mathcal{A}_i] \|_1 \, \gamma_q(E_i C[\mathcal{A}_i]) \\
&\le \gamma_q\Big(\sum_{k=i+1}^{N} E_k \, C[\mathcal{A}_i]\Big)
\end{aligned}$$

$$+ \gamma_q(E_i\, C[\mathcal{A}_i])$$
$$= \gamma_q(C[\mathcal{A}_i]).$$

Thus $\|K\|_1 \le \|C[\mathcal{A}_i]\|_1 < 1$ and $\|C[\mathcal{A}_{i+1}]\|_1 \le \|C[\mathcal{A}_i]\|_1$. By construction this implies $\|L[C_i]E_i\|_1 \le \|C[A]\|_1$ for $1 \le i \le N$. Now, let the blocks of A_i be a_{mp}, $1 \le m, p \le N$. Then $\mathcal{A}_{i+1} = (a_{mp} + c_{mi}a_{ip})_{i+1\le m,p\le N}$, and

$$\gamma_q(\mathcal{A}_{i+1}) \le \gamma_q\Big(\sum_{k=i+1}^{N} E_i\mathcal{A}_i\Big)$$
$$+ \left\|\begin{pmatrix} c_{i+1,i} \\ \vdots \\ c_{Ni} \end{pmatrix}\right\|_1 \gamma_q(E_i\mathcal{A}_i)$$
$$\le \gamma_q\Big(\sum_{k=i+1}^{N} E_k\mathcal{A}_i\Big) + \gamma_q(E_i\mathcal{A}_i)$$
$$= \gamma_q(\mathcal{A}_i),$$
$$\|\mathcal{A}_{i+1}\|_1 \le \|\mathcal{A}_i\|_1.$$

QED

We note that if A is symmetric then $C[A] = B[A]^T$ and Theorems 3.2 and 3.3 become nearly identical. Of course, B and C are always similar since $C = DBD^{-1}$. When A is positive definite, Cline [C3] has shown that the eigenvalues of $\mathcal{A}_i$ interlace those of $\mathcal{A}_{i-1}$. Actually, the proof depends on use of the Cholesky factorization, but the reduced matrices are the same in both methods. In consequence of Cline's result we have $\|\mathcal{A}_{i+1}\|_2 \le \|\mathcal{A}_i\|_2$.

The statements of Theorems 3.2 and 3.3 depend heavily on the particular norms that were used. It is not true in general, for example, that if $\|B[A_1]\|_1 < 1$ then $\|B[A_2]\|_1 \le \|B[A_1]\|_1$ or even $\|B[\mathcal{A}_2]\|_1 \le \|B[\mathcal{A}_1]\|_1$. This makes the extension to other norms quite difficult to determine.

3.A.2. Further Properties of Block Elimination

It is well-known that the block factorization $A = L\Delta U$ is given by

$$L = I - \sum_{j=1}^{N} L[C_j]E_j,$$

$$\Delta = D[A_N],$$

$$U = I - B_N = I - \sum_{j=1}^{N} E_j\, U[B_j].$$

We have therefore shown in Theorems 3.2 and 3.3 that if $\|C_1\|_1 < 1$ then $\|L\|_1 \le 1 + \|C_1\|_1 < 2$, $\|L\| \le 1 + Nn\|C_1\|_1$, $n = \max_j n_j$, and if $\|B_1\| < 1$ then $\|U\| \le 1 + \|B_1\| < 2$, $\|U\|_1 \le 1 + Nn\|B_1\|$. (For block tridiagonal matrices the factor N can be removed, so $\|L\| \le 1 + n\|C_1\|_1$, $\|U\|_1 \le 1 + n\|B_1\|$.) Moreover, the factorization can be completed without permuting the block rows of A. The purpose of block partial pivoting for arbitrary matrices is to force the inequality $\|L[C_j]E_j\| \le 1$. Broyden [B15] contains a number of interesting results related to pivoting in Gaussian elimination. In particular it is shown that partial pivoting has the effect of bounding the condition number of L, though the bound and condition number can actually be quite large in the general case. We will have some more comments on pivoting after looking at some matrix properties that are invariant under block elimination (cf. [B11]).

Corollary 3.2. Block elimination preserves strict diagonal dominance.

Proof. It follows from Theorem 3.2 that Gaussian elimination preserves strict diagonal dominance: simply partition A into 1×1 blocks. Now suppose we have A_i and are about to generate A_{i+1}. This can be done either by one stage of block elimination or by n_i stages of Gaussian elimination, for both processes produce the same result when using exact arithmetic [H8, p. 130]. Thus if A_i is strictly diagonally dominant, then so is A_{i+1}. QED

It is shown in [B4] that Gaussian elimination maps G-matrices into G-matrices. This can be extended in an obvious fashion to the case of block elimination. We also have

Corollary 3.3. Block elimination maps H-matrices into H-matrices.

Proof. Recall that A is an H-matrix if and only if there exists a nonsingular diagonal matrix E such that $A' = AE$ is strictly diagonally dominant. Consider the sequence

$$A'_1 = A',$$
$$A'_{i+1} = (I - L[A'_i]E_i\ D[A'_i]^{-1})A'_i.$$

Clearly $A'_1 = A_1E$; suppose that $A'_i = A_iE$ is strictly diagonally dominant. We then have

$$\begin{aligned} A'_{i+1} &= (I - L[A_i]E\ E_iE^{-1}D[A_i]^{-1})A_iE \\ &= (I - L[A_i]E_i\ D[A_i]^{-1})A_iE \\ &= A_{i+1}E, \end{aligned}$$

so A_{i+1} is an H-matrix since A'_{i+1} is strictly diagonally dominant by Corollary 3.2. QED

We have chosen to give this more concrete proof of Corollary 3.3 because it allows us to estimate another norm of B[A] when A is an H-matrix. Define $\|M\|_E = \|E^{-1}ME\|$. Then since $A' = AE$ implies $B[A'] = E^{-1}B[A]E$ it follows that if A is an H-matrix then $\|B[A_{i+1}]\|_E \le \|B[A_i]\|_E$. Since the matrix E will generally remain unknown this is in the nature of an existential proof.

The proof of Corollary 3.2 brings out an important interpretation of Theorem 3.2. If block elimination is actually performed then our result gives direct information about properties of the reduced matrices. If not, and ordinary Gaussian elimination is used instead, then Theorem 3.2 provides information about the process after each n_i steps. While we are not assured that pivoting will not be necessary, we do know that pivot selections for these n_i steps will be limited to elements within the d_i block for the block tridiagonal LU factorization. When A has been partitioned to deal with mass storage hierarchies this will be important knowledge (see Section 10.A).

One more consequence of Theorem 3.2 is

Corollary 3.4. In the block tridiagonal LU factorization, if $\|B_1\| < 1$ then $\text{cond}(d_j) = \|d_j\| \, \|d_j^{-1}\| < 2 \,\text{cond}(b_j) / (1 - \|B_1\|^2)$.

Proof. Letting $\sigma_j = b_j^{-1} a_j d_{j-1}^{-1} c_{j-1}$, we have $d_j = b_j(I - \sigma_j)$, $\|\sigma_j\| \le \|B_1\| \, \|B_N\| \le \|B_1\|^2 < 1$, so that $\|d_j\| \le \|b_j\| (1 + \|\sigma_j\|) < 2\|b_j\|$, and $\|d_j^{-1}\| \le \|b_j^{-1}\| \, \|(I - \sigma_j)^{-1}\| \le \|b_j^{-1}\| / (1 - \|\sigma_j\|) \le \|b_j^{-1}\| / (1 - \|B_1\|^2)$.

QED

It is important to have bounds on $\text{cond}(d_j)$ since systems involving d_j must be solved during the factorization. Of course, a posteriori bounds on $\|d_j^{-1}\|$ may be computed at a moderate cost. We note also that $\text{cond}(A) \le \max_j 2\,\text{cond}(b_j) / (1 - \|B_1\|)$ when $\|B_1\| < 1$, and it is entirely possible for the individual blocks b_j and d_j to be better-conditioned than A.

The inequalities in Theorem 3.2 are the best possible under the condition $\|B_1\| < 1$. This is because, for example, the first block rows of B_1 and B_2 are identical. However, it is possible to derive stronger results by using different assumptions about B_1.

Theorem 3.4. [H4] For $A = (a_j, b_j, c_j)$, let $\lambda(A) = \max_j(4\| b_j^{-1}a_j\| \|b_{j-1}^{-1}c_{j-1}\|)$, $e_j = b_j^{-1}d_j$. If $\lambda(A) \le 1$ then $\| I - e_j\| \le (1 - \sqrt{1-\lambda})/2$ and $\| e_j^{-1}\| \le 2/(1 + \sqrt{1-\lambda})$.

Some useful consequences of this theorem include better estimates for $\| B_N\|$ and a smaller upper bound on the condition numbers of the d_j's. For we have $B_N = (0,0,-d_j^{-1}c_j) = (0,0,-e_j^{-1}b_j^{-1}c_j)$, which implies that $\| B_N\| \le 2\| U[B_1]\| / (1 + \sqrt{1-\lambda})$. As an example, if $A = (1,3,1)$ then Theorem 3.2 predicts only $\| B_N\| \le \| B_1\| = 2/3$, while by Theorem 3.4 we obtain $\| B_N\| \le 2(1/3) / (1 + \sqrt{1-4/9}) \doteq 0.382$. Also, $\| d_j^{-1}\| \le \| d_j^{-1}b_j\| \|b_j^{-1}\| = \|e_j^{-1}\| \|b_j^{-1}\| \le 2\|b_j^{-1}\|$, and $\|d_j\| \le \|b_j - d_j\| + \|b_j\| = \|b_j(I - e_j)\| + \|b_j\| \le \| b_j\| \| I - e_j\| + \| b_j\| \le 3\| b_j\|/2$. It follows that $\text{cond}(d_j) = \| d_j^{-1}\| \|d_j\| \le 3\,\text{cond}(b_j)$. More precisely, $\text{cond}(d_j) \le (1 + 2\mu(A))\ \text{cond}(b_j)$, $\mu = (1 - \sqrt{1-\lambda})/(1 + \sqrt{1-\lambda})$. This is further proof of the stability of the block LU factorization.

Theorem 3.4 has been used in the analysis of an iteration to compute the block diagonal matrix $(0,d_j,0)_N$ [H4]. Such methods are potentially useful in the treatment of time-dependent problems, where good initial approximations are available. This particular iteration is designed for use on a parallel computer and has several remarkable properties; for details, see [H4] and Section 8.E.

Two results on the block LU factorization for block tridiagonal matrices were previously obtained by Varah.

Theorem 3.5. [V1] If A is block diagonally dominant $(\| b_j^{-1}\|(\|a_j\| + \|c_j\|) \le 1)$ and $\| c_j\| \ne 0$, then $\| d_j^{-1}\| \le \| c_j\|^{-1}$, $\| \ell_j\| \le \| a_j\| / \| c_{j-1}\|$, and $\| d_j\| \le \| b_j\| + \| a_j\|$.

In comparison to Theorem 3.2, the bound on $\|d_j^{-1}\|$ corresponds to the bound $\|d_j^{-1}c_j\| < 1$, which follows from $\|B_N\| < 1$, and the bound on $\|d_j\|$ corresponds to the bound $\|A_N\| \le \|A\|$. In contrast to Theorem 3.2, $\|B_1\| < 1$ only allows us to conclude $\|d_j^{-1}\| \le \|b_j^{-1}\| / (1 - \|B_1\|^2)$, and we can only estimate $\|\ell_j\|$ in the weak form $\|\ell_j c_{j-1}\| \le \|a_j\| \, \|d_{j-1}^{-1}c_{j-1}\| \le \|a_j\|$.

Varah's second result is

Theorem 3.6. [V1] Let $\alpha_j = (\|b_j^{-1}a_j\| \, \|b_{j-1}^{-1}c_{j-1}\|)^{1/2}$, $2 \le j \le N$, and suppose $\alpha_j \ne 0$. Then the block LU factorization of A is numerically stable (i.e., there exists a constant K(A) such that $\max(\|\ell_j\|, \|d_j\|) \le K(A)$) if $(\alpha_j, 1, \alpha_{j-1})$ is positive semidefinite.

We note that $(\alpha_j, 1, \alpha_{j-1})$ is a symmetrization of $(\|b_j^{-1}a_j\|, 1, \|b_j^{-1}c_j\|)$ and that $\lambda(A) \le 1$ implies that it is positive definite. The actual bounds on $\|\ell_j\|$, $\|d_j\|$ given by Varah [V1] for this case will not be important for this discussion.

Since many block tridiagonal systems arise from the discretization of differential equations, a number of results have been obtained under assumptions closely related to the original continuous problem. Rosser [R7] shows that if the system $Ax = v$, $A = (a_j, b_j, c_j)$, satisfies a certain maximum principle then A is nonsingular, the LU factorization exists (each d_j is nonsingular), B_N is non-negative and $\|B_N\| \le 1$. Improved bounds on B_N are obtained when A is defined by the nine-point difference approximation to Poisson's equation. In particular there is the remarkable observation that the spectral condition number of d_j is less than 17/3. Such results depend strongly on the particular matrix under consideration. For examples of related work, see [B18], [C1], [D2].

3.A.3. Back Substitution

So far we have only considered the elimination phase of the solution of $Ax = v$. For block tridiagonal matrices $A = (a_j, b_j, c_j)_N$ this consists of the process

$$d_1 = b_1,\ f_1 = v_1,$$
$$d_j = b_j - a_j d_{j-1}^{-1} c_{j-1},\ f_j = v_j - a_j d_{j-1}^{-1} f_{j-1},$$
$$j = 2,\dots,N.$$

The back substitution phase consists of the process

$$x_N = d_N^{-1} f_N,$$
$$x_j = d_j^{-1}(f_j - c_j x_{j+1}),\ j = N-1,\dots,1.$$

In matrix terms these are the processes

$$A_1 = A,\ f^{(1)} = v,$$
$$A_{i+1} = (I + L[C_i]E_i)A_i,\quad i = 1,\dots,N-1,$$
$$f^{(i+1)} = (I + L[C_i]E_i)f^{(i)},\ i = 1,\dots,N-1,$$
$$g = x^{(N)} = D[A_N]^{-1} f^{(N)},$$
$$x^{(i)} = g + E_i B_N\, x^{(i+1)},\quad i = N-1,\dots,1,$$
$$x = x^{(1)}.$$

An alternate form of back substitution is

$$x^{(N)} = D[A_N]^{-1} f^{(N)},$$
$$x^{(i)} = (I + B_N E_{i+1}) x^{(i+1)},\quad i = N-1,\dots,1,$$
$$x = x^{(1)}.$$

In either case it is seen that B_N plays an essential role in studying the back-substitution process.

For those situations where a band elimination method for $Ax = v$ might be preferred, the matrix B_N maintains its importance for back substitution. We have $A = (a_j, b_j, c_j) = (\hat{a}_j, \hat{\ell}_j, 0)(0, \hat{u}_j, \hat{c}_j)$, $d_j = \hat{\ell}_j \hat{u}_j$, $\hat{a}_j = a_j \hat{u}_{j-1}^{-1}$, $\hat{c}_j = \hat{\ell}_j^{-1} c_j$, where $\hat{\ell}_j (\hat{u}_j)$ is lower (upper) triangular. The elimination phase generates vectors $\hat{f}_j = \hat{\ell}_j f_j$, and the back substitution computes $x_N = \hat{u}_N^{-1} \hat{f}_N = d_N^{-1} f_N$, $x_j = \hat{u}_j^{-1}(\hat{f}_j - \hat{c}_j x_{j+1}) = d_j^{-1}(f_j - c_j x_{j+1})$. Except for the specific form of roundoff errors the band and block back substitutions must behave identically.

Let us briefly consider the effect of rounding errors in back substitution; the computed solution will be $y_j = x_j + \xi_j$ and we require bounds on ξ_j. Suppose the computed forward elimination yields $d_j^* = d_j + \delta_j^{(1)}$, $f_j^* = f_j + \varphi_j$. Let η_j represent the errors introduced in forming $f_j^* - c_j y_{j+1}$, so that y_j satisfies $(d_j^* + \delta_j^{(2)}) y_j = f_j^* - c_j y_{j+1} + \eta_j$ if ordinary Gaussian elimination is used to solve for y_j in $d_j^* y_j = f_j^* - c_j y_{j+1} + \eta_j$. It follows that $y_j = d_j^{-1}(f_j - c_j y_{j+1}) + \epsilon_j$, where $\epsilon_j = d_j^{-1}(\varphi_j + \eta_j - (\delta_j^{(1)} + \delta_j^{(2)}) y_j)$, which implies $\xi_j = \epsilon_j - d_j^{-1} c_j \xi_{j+1}$. Thus $\| \xi_j \| \le \| B_N \| \, \| \xi_{j+1} \| + \| \epsilon_j \|$, and the previous errors tend to be damped out in later computations if $\| B_N \| < 1$. If we have a uniform upper bound for the ϵ_j's, $\| \epsilon_j \| \le \epsilon$, and $\| B_N \| < 1$, then we obtain $\| \xi_j \| \le (N - j + 1)\epsilon$, $\| \xi_j \| \le (1 - \| B_N \|^{N-j+1})\epsilon/(1 - \| B_N \|) < \epsilon/(1 - \| B_N \|)$.

3.A.4. Special Techniques

In some cases the matrix $A = (a_j, b_j, c_j)$ possesses special properties that may be important to preserve in order to satisfy a strong stability criterion. For example, we might have $b_j = t_j - a_j - c_j$ where $\| t_j \|$ is small or zero; this requires that all blocks be $n \times n$. Scalar tridiagonal systems with this property arise from the discretization of second order ODE boundary value problems. Babuska [B1-3] has discussed a combined LU - UL factorization for such a matrix which allows a very effective backward error analysis to the ODE and a stable numerical solution of a difficult singular perturbation problem posed by Dorr [D3]. (See [H7] for a symbolic approach to this problem.)

There are two essential features to Babuska's method. The first is to express b_j and d_j indirectly by starting with t_j and computing the sequence $s_1 = t_1$, $s_j = t_j - a_j d_{j-1}^{-1} s_{j-1}$. By this we have $d_j = s_j - c_j$. The second feature is to augment the forward elimination with a backward elimination and dispense with the back substitution by combining the results of elimination. Specifically, suppose $A = (-p_j, t_j + p_j + q_j, -q_j)_N$, $p_j, q_j, t_j \geq 0$, $p_1 = q_N = 0$, A nonsingular. The algorithm for $Ax = v$ is

$$s_1 = t_1,\ f_1 = v_1,\ s_N^* = t_N,\ f_N^* = v_N,$$

$$\left.\begin{aligned} s_j &= t_j + p_j(s_{j-1} + q_{j-1})^{-1} s_{j-1} \\ f_j &= v_j + p_j(s_{j-1} + q_{j-1})^{-1} f_{j-1} \end{aligned}\right\}\quad j = 2,\ldots,N,$$

$$\left.\begin{aligned} s_j^* &= t_j + q_j(s_{j+1}^* + p_{j+1})^{-1} s_{j+1}^* \\ f_j^* &= v_j + q_j(s_{j+1}^* + p_{j+1})^{-1} f_{j+1}^* \end{aligned}\right\}\quad j = N-1,\ldots,1,$$

$$x_j = (s_j + s_j^* - t_j)^{-1}(f_j + f_j^* - v_j),\quad j = 1,\ldots,N.$$

In matrix terms, we factor $A = A_L + A_D + A_U = (I + L)(D + A_U) = (I + U^*)(D^* + A_L)$ and solve $(I + L)f = v$, $(I + U^*)f^* = v$. Adding $(D + A_U)x = f$, $(D^* + A_L)x = f^*$ and $-A_D x = -A_D x$ we obtain $(D + D^* - A_D)x = f + f^* - Ax = f + f^* - v$.

Babuska's algorithm has some unusual round-off properties (cf. [M3]) and underflow is a problem though correctable by scaling. It handles small row sums by taking account of dependent error distributions in the data, and thus fits into Kahan's notion that "conserving confluence curbs ill-condition" [K1].

To see how the method fits in with our results, we note that, for $n = 1$, since $p_j, q_j, t_j \geq 0$ we have $s_1 \geq 0$, $s_j = t_j + p_j d_{j-1}^{-1} s_{j-1} \geq t_j$. Thus the row sums can only increase during the factorization. On the other hand, they will not increase much since $s_j = t_j + p_j(s_{j-1} + q_{j-1})^{-1} s_{j-1} \leq t_j + p_j$. These are analogues of the results in Theorem 3.2.

3.B. Gauss-Jordan Elimination

An algorithm closely related to block elimination is the block Gauss-Jordan elimination

$$\bar{A}_1 = A,$$
$$\bar{A}_{i+1} = (I + C[\bar{A}_i]E_i)\bar{A}_i, \quad i = 1,\ldots,N.$$

This avoids back substitution in the solution of $Ax = v$ since $\bar{A}_{N+1}$ is block diagonal. The rest of the Gauss-Jordan process is

$$\bar{f}^{(1)} = v,$$
$$\bar{f}^{(i+1)} = (I + C[\bar{A}_i]E_i)\bar{f}^{(i)}, \quad i = 1,\ldots,N,$$
$$x = \bar{A}_{N+1}^{-1}\,\bar{f}^{(N+1)}.$$

<u>Theorem 3.7.</u> Let $\bar{B}_i = B[\bar{A}_i]$. If $\|\bar{B}_1\| < 1$ then $\|\bar{A}_{i+1}\| \leq \|\bar{A}_i\|$ and $\|\bar{B}_{i+1}\| \leq \|\bar{B}_i\|$ for $i = 1,\ldots,N$.

<u>Sketch of Proof</u>. The proof parallels that of Theorem 3.2. Note $\bar{B}_1 = B_1$; suppose $\|\bar{B}_i\| < 1$. Let $F_i = L[\bar{A}_i] + U[\bar{A}_i]$, so that $\bar{A}_{i+1} = (I - F_i E_i D[\bar{A}_i]^{-1})\bar{A}_i = \bar{A}_i - F_i E_i + F_i E_i \bar{B}_i$. This will show the first part. Now define $Q = D[\bar{A}_i]^{-1} F_i E_i D[\bar{A}_i]^{-1} \bar{A}_i$, so that $\bar{A}_{i+1} = \bar{A}_i - D[\bar{A}_i]Q$, $D[\bar{A}_{i+1}] = D[\bar{A}_i](I - S)$, $S = D[Q]$. We have $Q = (-\bar{B}_i E_i)(I - \bar{B}_i) = -\bar{B}_i E_i + \bar{B}_i E_i \bar{B}_i$, so $S = D[\bar{B}_i E_i \bar{B}_i]$ and $\|S\| \le \|\bar{B}_i\|^2 < 1$. Also, $(I - S)\bar{B}_{i+1} = \bar{B}_i - S + Q$. Let $T = \bar{B}_i - S + Q$, $J = S + T$, so that $|J| = |S| + |T|$, and Lemma 2.1 applies if $\|J\| < 1$. But $J = \bar{B}_i(I - E_i + E_i \bar{B}_i)$, so $\|J\| \le \|\bar{B}_i\|$. From Lemma 2.1 we conclude $\|\bar{B}_{i+1}\| \le \|J\|$. QED

When applied to block tridiagonal matrices the block Gauss-Jordan algorithm requires many more arithmetic operations than the block LU factorization, and cannot be recommended on this basis. However, it has some numerical properties that foreshadow our analysis of the more important odd-even reduction algorithm.

The Gauss-Jordan elimination on block tridiagonals may be expressed in the following way:

```
$d_1 = b_1$
for $j = 1,\dots,N-1$
    $u_j = -d_j^{-1} c_j$
    $d_{j+1} = b_{j+1} + a_{j+1} u_j$
    for $k = 1,\dots,j-1$
        $e_{k,j+1} = e_{kj} u_j$
    $e_{j,j+1} = c_j$
```

In this form the matrix $\bar{A}_j$ is

$$\begin{pmatrix}
d_1 & & & & & e_{1j} & & & \\
 & d_2 & & & & e_{2j} & & & \\
 & & d_3 & & & e_{3j} & & & \\
 & & & \ddots & & \vdots & & & \\
 & & & & d_{j-1} & e_{j-1,j} & & & \\
 & & & & 0 & d_j & c_j & & \\
 & & & & & a_{j+1} & b_{j+1} & c_{j+1} & \\
 & & & & & & \ddots & \ddots & \ddots \\
 & & & & & & & a_N & b_N
\end{pmatrix},$$

with $\bar{A}_{N+1} = \mathrm{diag}(d_1, d_2, \ldots, d_N) = D[A_N]$, $\bar{B}_{N+1} = 0$.

We observe that, for $1 \leq j < j + k \leq N$, $-d_j^{-1} e_{j,j+k} = u_j u_{j+1} \cdots u_{j+k-1}$, which is the $(j, j + k)$ block of B_N^k. Thus we are able to refine Theorem 3.7 by giving closer bounds on the first i-1 block rows of $\bar{B}_i$.

<u>Corollary 3.5.</u> If $\|B_1\| < 1$ and (point) row ℓ is contained in block row j with $1 \leq j \leq i-1$, then

$$\rho_\ell(\bar{B}_i) \leq \|-d_j^{-1} e_{ji}\| \leq \|B_N^{i-j}\|.$$

If (point) row ℓ is contained in block row j with $i \leq j \leq N$, then

$$\rho_\ell(\bar{B}_i) = \rho_\ell(B_i) \leq \|B_i\|.$$

This is our first result on the convergence to zero of portions of a matrix during the reduction to a block diagonal matrix. It also provides a rather different situation from Gauss-Jordan elimination with partial pivoting applied to an arbitrary matrix [P1]. In that case the multipliers above the main diagonal can be quite large, while we have seen for block tridiagonal matrices with $\|B_1\| < 1$ that the multipliers actually decrease in size.

3.C. Parallel Computation

We close this section with some remarks on the applications and inherent parallelism of the methods; see also [H3]. When a small number of parallel processors are available the block and band LU factorizations are natural approaches. This is because the solution (or reduction to triangular form) of the block system $d_j u_j = -c_j$ is naturally expressed in terms of vector operations on rows of the augmented matrix $(d_j | -c_j)$, and these vector operations can be executed efficiently in parallel. However, we often cannot make efficient use of a pipeline processor since the vector lengths might not be long enough to overcome the vector startup costs. Our comparison of algorithms for a pipeline processor (Section 10.C) will therefore consider the LU factorization executed in scalar mode. We also note that the final stage of Babuska's LU-UL factorization is the solution of a block diagonal system, which is entirely parallel.

On the other hand, the storage problems created by fill-in with these methods when the blocks are originally sparse are well-known. For large systems derived from elliptic equations this is often enough to disqualify general elimination methods as competitive algorithms. The block elimination methods will be useful when the block sizes are small enough so that they can be represented directly as full matrices, or when a stable implicit representation can be used instead.

4. QUADRATIC METHODS

In Section 3 we have outlined some properties of the block LU factorization and block Gauss-Jordan elimination. The properties are characterized by the generation of a sequence of matrices M_i such that $\| B[M_{i+1}] \| \le \| B[M_i] \|$. From this inequality we adopted the name "linear methods". Now we consider some quadratic methods, which are characterized by the inequality $\| B[M_{i+1}] \| \le \| B[M_i]^2 \|$.

There are two basic quadratic methods, odd-even elimination and odd-even reduction. The latter usually goes under the name cyclic reduction, and can be regarded as a compact version of the former. We will show that odd-even elimination is a variation of the quadratic Newton iteration for A^{-1}, while odd-even reduction is a special case of the more general cyclic reduction technique of Hageman and Varga [H1] and is equivalent to block elimination on a permuted system. Both methods are ideally suited for use on a parallel computer, as many of the quantities involved may be computed independently of the others [H3]. Semidirect methods based on the quadratic properties are discussed in Section 7.

4.A. Odd-Even Elimination

We first consider odd-even elimination. Pick three consecutive block equations from the system $Ax = v$, $A = (a_j, b_j, c_j)$:

$$\begin{aligned} a_{k-1}x_{k-2} + b_{k-1}x_{k-1} + c_{k-1}x_k &= v_{k-1} \\ a_k x_{k-1} + b_k x_k + c_k x_{k+1} &= v_k \\ a_{k+1}x_k + b_{k+1}x_{k+1} + c_{k+1}x_{k+2} &= v_{k+1} \end{aligned} \tag{4.1}$$

If we multiply equation k-1 by $-a_k b_{k-1}^{-1}$, equation k+1 by $-c_k b_{k+1}^{-1}$, and add, the result is

$$
\begin{aligned}
&(-a_k b_{k-1}^{-1} a_{k-1}) x_{k-2} \\
&+ (b_k - a_k b_{k-1}^{-1} c_{k-1} - c_k b_{k+1}^{-1} a_{k+1}) x_k \\
(4.2) \qquad &+ (-c_k b_{k+1}^{-1} c_{k+1}) x_{k+2} \\
&= (v_k - a_k b_{k-1}^{-1} v_{k-1} - c_k b_{k+1}^{-1} v_{k+1}).
\end{aligned}
$$

For $k = 1$ or N there are only two equations involved, and the necessary modifications should be obvious. The name odd-even comes from the fact that if k is even then the new equation has eliminated the odd unknowns and left the even ones. This is not the only possible row operation to achieve this effect, but others differ only by scaling and possible use of a matrix commutativity condition.

By collecting the new equations into the block pentadiagonal system $H_2 x = v^{(2)}$, it is seen that the row eliminations have preserved the fact that the matrix has only three non-zero diagonals, but they now are farther apart. A similar set of operations may again be applied, combining equations $k-2$, k and $k+2$ of H_2 to produce a new equation involving the unknowns x_{k-4}, x_k and x_{k+4}. These new equations form the system $H_3 x = v^{(3)}$.

The transformations from one system to the next may be succinctly expressed in the following way. Let $H_1 = A$, $v^{(1)} = v$, $m = \lceil \log_2 N \rceil$, and define

$$
\begin{aligned}
H_{i+1} &= (I + C[H_i])H_i, \\
v^{(i+1)} &= (I + C[H_i])v^{(i)}, \qquad i = 1,2,\ldots,m.
\end{aligned}
$$

Block diagrams of the H sequence for $N = 15$ are shown in Figure 4.1a. The distance between the three non-zero diagonals doubles at each stage until the block diagonal matrix H_{m+1} is obtained; the solution x is then obtained by $x = H_{m+1}^{-1} v^{(m+1)}$.

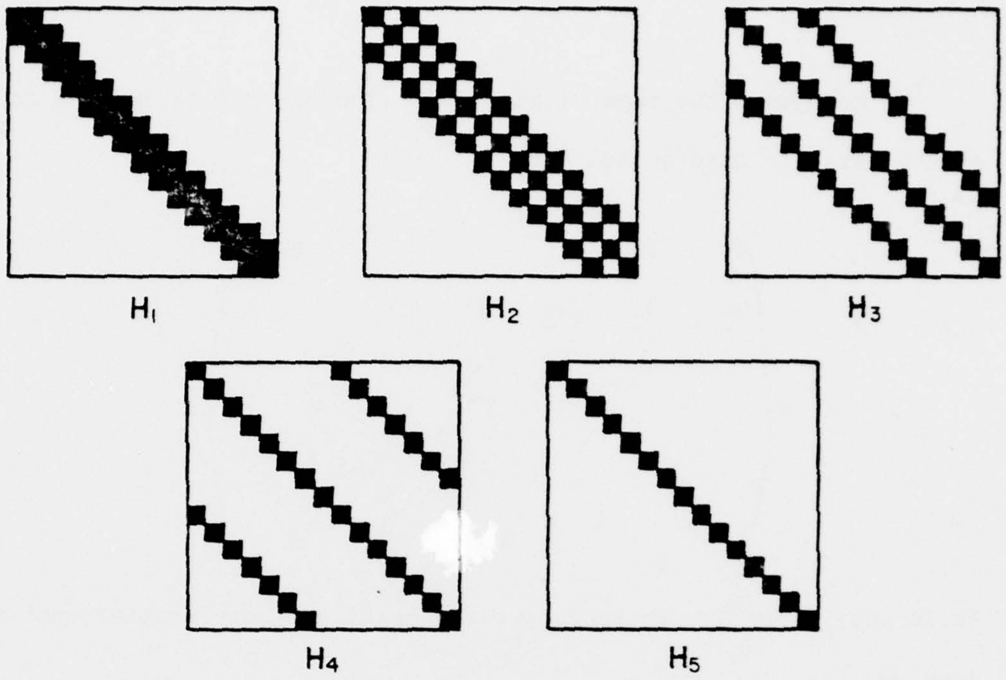


Figure 4.1a. H_i, $i=1,\ldots,m+1$; $N = 15$

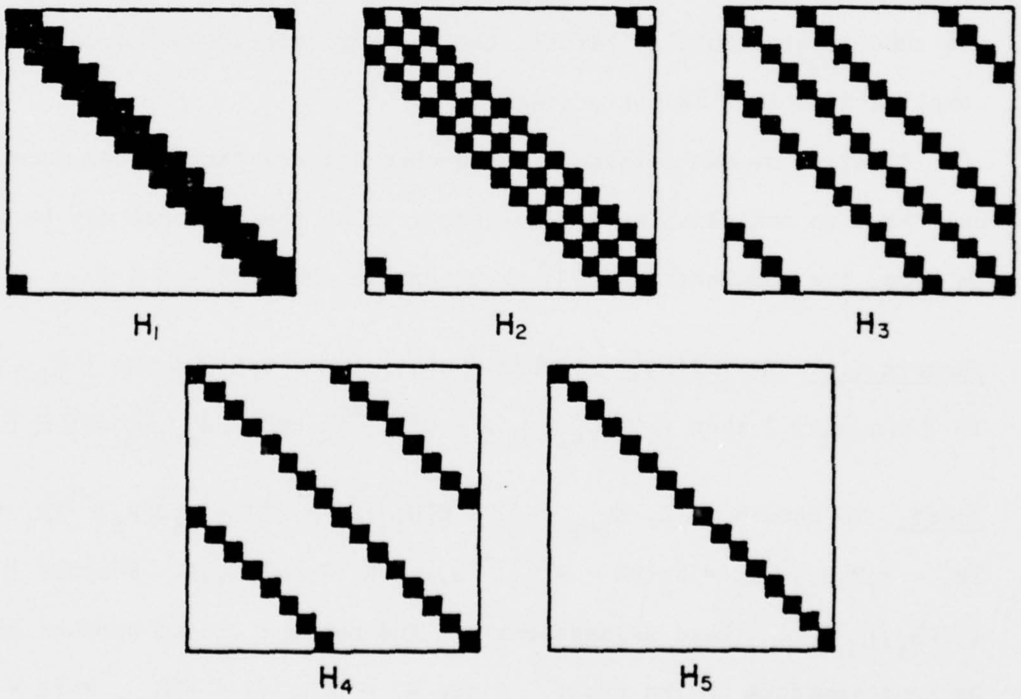


Figure 4.1b. H_i, $i=1,\ldots,m+1$; $N = 16$

We note that the same elimination principle may be applied to matrices of the periodic tridiagonal form

$$A^* = \begin{pmatrix} b_1 & c_1 & & & & a_1 \\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{N-1} & b_{N-1} & c_{N-1} \\ c_N & & & & a_N & b_N \end{pmatrix}.$$

It is only necessary to include row operations that "wrap around" the matrix, such as

$$\text{row}(N) - a_N b_{N-1}^{-1} \text{ row}(N-1) - c_N b_1^{-1} \text{ row}(1).$$

An H sequence for N = 16 is diagrammed in Figure 4.1b, and is constructed by the same general rule as before, though there are serious and interesting complications if N is not a power of 2.

Clearly, we can define an H sequence for any partitioned matrix A, and only have to establish conditions under which the sequence may be continued. In fact, the sequence $\|B[H_i]\|$ is quadratic when $\|B[A]\| < 1$.

Theorem 4.1. If $\|B[A]\| < 1$ then $\|B[H_{i+1}]\| \le \|B[H_i]^2\|$ and $\|H_{i+1}\| \le \|H_i\|$. If $\|C[A]\|_1 \le 1$ then $\|C[H_{i+1}]\|_1 \le \|C[H_i]^2\|_1$ and $\|H_{i+1}\|_1 \le \|H_i\|_1$.

Proof. We have $H_1 = A$, $H_{i+1} = (I + C[H_i])H_i = (2I - H_i D[H_i]^{-1})H_i = 2H_i - H_i D[H_i]^{-1} H_i = H_i(2I - D[H_i]^{-1}H_i) = H_i(I + B[H_i])$. Suppose $\|B[H_i]\| < 1$, $\|C[H_i]\|_1 < 1$. These assumptions are independent of one another and will be kept separate in the proof. Since $H_i = D[H_i](I - B[H_i]) = (I - C[H_i])D[H_i]$, we have $H_{i+1} = D[H_i](I - B[H_i]^2) = (I - C[H_i]^2)D[H_i]$. Setting $S = D[B[H_i]^2]$,

$T = D[C[H_i]^2]$, we obtain $D[H_{i+1}] = D[H_i](I - S) = (I - T)D[H_i]$. Since $\|S\| \le \|B[H_i]^2\| < 1$, $\|T\|_1 \le \|C[H_i]^2\|_1 < 1$, and $D[H_i]$ is invertible, it follows that $D[H_{i+1}]$ is invertible. We have $B[H_{i+1}] = I - D[H_{i+1}]^{-1}H_{i+1} = I - (I - S)^{-1}D[H_i]^{-1}D[H_i](I - B[H_i]^2)$, so $B[H_i]^2 = (I - S)B[H_{i+1}] + S$; similarly $C[H_i]^2 = C[H_{i+1}](I - T) + T$. Now, S is block diagonal and the block diagonal part of $B[H_{i+1}]$ is null, so $|B[H_i]^2| = |(I - S)B[H_{i+1}]| + |S|$. Lemma 2.2 now applies, and we conclude $\|B[H_{i+1}]\| \le \|B[H_i]^2\|$. Lemma 2.4 implies that $\|C[H_{i+1}]\|_1 \le \|C[H_i]^2\|_1$. To show $\|H_{i+1}\| \le \|H_i\|$ if $\|B[H_i]\| < 1$, write $H_{i+1} = D[H_i](I - D[H_i]^{-1}(D[H_i] - H_i)B[H_i]) = D[H_i] + (H_i - D[H_i])B[H_i]$. We then have, for $1 \le \ell \le \sum_{j=1}^{N} n_j$,

$$\begin{aligned} \rho_\ell(H_{i+1}) &\le \rho_\ell(D[H_i]) + \rho_\ell(H_i - D[H_i])\,\|B[H_i]\| \\ &\le \rho_\ell(D[H_i]) + \rho_\ell(H_i - D[H_i]) \\ &= \rho_\ell(H_i). \end{aligned}$$

To show $\|H_{i+1}\|_1 \le \|H_i\|_1$ if $\|C[H_i]\|_1 < 1$, write $H_{i+1} = (I - C[H_i](D[H_i] - H_i) \times D[H_i]^{-1})D[H_i] = D[H_i] + C[H_i](H_i - D[H_i])$. We have

$$\begin{aligned} \gamma_\ell(H_{i+1}) &\le \gamma_\ell(D[H_i]) + \|C[H_i]\|\,\gamma_\ell(H_i - D[H_i]) \\ &\le \gamma_\ell(D[H_i]) + \gamma_\ell(H_i - D[H_i]) \\ &= \gamma_\ell(H_i). \end{aligned}$$

QED

For a block tridiagonal matrix A we have $B[H_{m+1}] = 0$ since H_{m+1} is block diagonal; thus the process stops at this point. The proof of Theorem 4.1 implicitly uses the convexity-like formulae

$$\begin{aligned} H_{i+1} &= H_i\, B[H_i] + D[H_i](I - B[H_i]) \\ &= C[H_i]H_i + (I - C[H_i])D[H_i]. \end{aligned}$$

As with Theorems 3.2 and 3.3, it is not difficult to find counterexamples to show that Theorem 4.1 does not hold for arbitrary matrix norms. In particular, it is not true that $\| B[A] \|_2 < 1$ implies $\| B[H_2] \|_2 \le \| B[H_1]^2 \|_2$ or even that $\| B[H_2] \|_2 < 1$.

The conclusions of Theorem 4.1 are that the matrix elements, considered as a whole rather than individually, do not increase in size, and that the off-diagonal blocks decrease in size relative to the diagonal blocks if $\| B[A] \| < 1$. The rate of decrease is quadratic, though if $\| B[A] \|$ is very close to 1 the actual decreases may be quite small initially. The relative measure quantifies the diagonal dominance of H_i, and shows that the diagonal elements will always be relatively larger than the off-diagonals. One conclusion that is not reached is that they will be large on an absolute scale. Indeed, consider the matrix $A = (-1, 2+\epsilon, -1)$, for which the non-zero diagonals of H_i are

$$(-1, 2+\epsilon_i, -1) \;/\; \Pi_{k=1}^{i-1}(2+\epsilon_k),$$
$$\epsilon_1 = \epsilon, \quad \epsilon_{k+1} = 4\epsilon_k + \epsilon_k^2 .$$

For small ϵ the off-diagonal elements will be about 2^{1-i} and the main diagonal elements about 2^{2-i} in magnitude. While not a serious decrease in size, periodic rescaling of the main diagonal to 1 would perhaps be desirable.

As already hinted in the above remarks, a more refined estimate of $\| B[H_{i+1}] \|$ may be made when dealing with tridiagonal matrices with constant diagonals [S2]. Let $A = (a,b,a)$, so that

$$H_2 = \begin{pmatrix} b-a^2/b & 0 & -a^2 & & & \\ 0 & b-2a^2/b & 0 & -a^2 & & \\ -a^2/b & 0 & b-2a^2/b & 0 & -a^2 & \\ & \ddots & \ddots & \ddots & \ddots & \ddots \\ & & -a^2/b & 0 & b-2a^2/b & 0 \\ & & & -a^2/b & 0 & b^2-a^2/b \end{pmatrix}.$$

It follows that when $2|a| < |b|$, $\|B[H_2]\| = 2|a^2/b|/|b-2a^2/b| = \|B[H_1]^2\| / (2 - \|B[H_1]^2\|)$, and thus $\frac{1}{2}\|B[H_1]^2\| < \|B[H_2]\| < \|B[H_1]^2\|$. Note that H_2 does not have constant diagonals, nor will any of the other H matrices, but most of the rows of H_2 are essentially identical, and it is these rows which determine $\|B[H_2]\|$. This property will be important for the odd-even reduction algorithms.

When $A = (a_j, b_j, c_j)$ is block diagonally dominant, we can write $H_i = (a_j^{(i)}, 0, \ldots, 0, b_j^{(i)}, 0, \ldots, 0, c_j^{(i)})$, $\beta_j^{(i)} = \|(b_j^{(i)})^{-1}\|(\|a_j^{(i)}\| + \|c_j^{(i)}\|)$, $\beta^{(i)} = \max_j \beta_j^{(i)}$. Without proving it here, we can show that if $\beta^{(1)} < 1$ then $\beta^{(i+1)} \le \beta^{(i)2}$. An irreducibility condition plus $\beta^{(1)} \le 1$ yields the same conclusion.

We also note that if A is symmetric then so is H_i, $i \ge 1$, and thus $\|B[H_i]\| = \|C[H_i]\|_1$.

Theorem 4.2. If A is block tridiagonal and positive definite then so is H_i, $i \ge 1$, and $\rho(B[H_i]) < 1$.

Proof. First we observe that each H_i is a 2-cyclic matrix [V3] since it can be permuted into block tridiagonal form. Suppose that H_i is positive definite. Since $D_i = D[H_i]$ is also positive definite, there exists a positive definite matrix E_i such that $D_i = E_i^2$. Let $\hat{H}_i = E_i^{-1} H_i E_i^{-1}$, so $\hat{H}_{i+1} = E_i^{-1} H_{i+1} E_i^{-1} = 2\hat{H}_i - \hat{H}_i^2$;

$\hat{H}_i$ and $\hat{H}_{i+1}$ are positive definite iff H_i and H_{i+1} are positive definite. Since $\hat{H}_i$ is a positive definite 2-cyclic matrix, we know from Corollary 2 of Theorem 3.6 and Corollary 1 of Theorem 4.3 of [V3] that $\rho(\hat{B}_i) < 1$, $\hat{B}_i = I - \hat{H}_i$. Since $B_i = I - D_i^{-1}H_i = E_i^{-1}\hat{B}_i E_i$, it follows that $\rho(B_i) < 1$ also. Let λ be an eigenvalue of $\hat{H}_i$. Then $-1 < 1-\lambda < 1$ since $\rho(\hat{B}_i) < 1$, and so we have $0 < \lambda < 2$ and $0 < 2\lambda - \lambda^2 \leq 1$. But $2\lambda - \lambda^2$ is an eigenvalue of $\hat{H}_{i+1} = 2\hat{H}_i - H_i^2$, which is therefore positive definite. QED

Any quadratic convergence result such as Theorem 4.1 calls to mind the Newton iteration. If $\mathfrak{N}(Y,Z) = Y(2I - ZY) = (2I - YZ)Y$, $Y^{(0)}$ is any matrix with $\| I - ZY^{(0)} \|_\alpha < 1$ for some norm $\| \cdot \|_\alpha$, and $Y^{(i+1)} = \mathfrak{N}(Y^{(i)}, Z)$, then $Y^{(i)}$ converges to Z^{-1} if it exists and $(I - ZY^{(i+1)}) = (I - ZY^{(i)})^2$ [H8]. It may be observed that $H_{i+1} = \mathfrak{N}(H_i, D[H_i]^{-1})$, so while the H sequence is not a Newton sequence there is a close relation between the two. Nor is it entirely unexpected that Newton's method should appear, cf. Yun [Y1] and Kung [K2] for other occurrences of Newton's method in algebraic computations. Kung and Traub [K3] give a powerful unification of such techniques.

The Newton iteration $Y^{(i+1)} = \mathfrak{N}(Y^{(i)}, Z)$ is effective because, in a certain sense, it takes aim on the matrix Z^{-1} and always proceeds towards it. We have seen that odd-even elimination applied to a block tridiagonal matrix eventually yields a block diagonal matrix. The general iteration $H_{i+1} = \mathfrak{N}(H_i, D[H_i]^{-1})$ incorporates this idea by aiming the ith step towards the block diagonal matrix $D[H_i]$. Changing the goal of the iteration at each step might be thought to destroy quadratic convergence, but we have seen that it still holds in the sequence $\| B[H_i] \|$.

4.B. Variants of Odd-Even Elimination

There are a number of variations of odd-even elimination, mostly discussed originally in terms of the odd-even reduction. For constant block diagonals $A = (a,b,a)$ with $ab = ba$, Hockney [H5] multiplies equation k-1 by $-a$, equation k by b, equation k+1 by $-a$, and adds to obtain

$$(-a^2)x_{k-2} + (b^2 - 2a^2)x_k + (-a^2)x_{k+2} = (bv_k - av_{k-1} - av_{k+1}).$$

This formulation is prone to numerical instability when applied to the discrete form of the Poisson equation, and a stabilization has been given by Buneman (see [B20] and the discussion below).

With Hockney's use of Fourier analysis the block tridiagonal system $Ax = v$ is transformed into a collection of tridiagonal systems of the form $(-1,\lambda,-1)(s_j) = (t_j)$, where $\lambda \geq 2$ and often $\lambda \gg 2$. It was Hockney's observation that the reduction, which generates a sequence of Toeplitz tridiagonal matrices $(-1,\lambda^{(i)},-1)$, could be stopped when $1/\lambda^{(i)}$ fell below the machine precision. Then the tridiagonal system was essentially diagonal and could be solved as such without damage to the solution of the Poisson equation.

Stone [S2], in recommending a variation of cyclic reduction for the parallel solution of general tridiagonal systems, proved quadratic convergence of the ratios $1/\lambda^{(i)}$ and thus of the B matrices for the constant diagonal case. Jordan [J1], independently of our own work [H2], also extended Stone's result to general tridiagonals and the odd-even elimination algorithm. The results given here apply to general block systems, and are not restricted solely to the block tridiagonal case, though this will be the major application.

We now show that quadratic convergence results hold for other formulations of the odd-even elimination; we will see later how these results carry over to odd-even reduction. In order to subsume several algorithms into one, consider the sequences $\bar{H}_1 = A$, $\bar{v}^{(1)} = v$,

$$\bar{H}_{i+1} = F_i(2D[\bar{H}_i] - \bar{H}_i)G_i\bar{H}_i,$$
$$\bar{v}^{(i+1)} = F_i(2D[\bar{H}_i] - H_i)G_i\bar{v}^{(i)},$$

where F_i, G_i are nonsingular block diagonal matrices. The sequence H_i corresponds to the choices $F_i = I$, $G_i = D[H_i]^{-1}$.

In the Hockney-Buneman algorithm for Poisson's equation with Dirichlet boundary conditions on a rectangle we have

$$A = (a,b,a),\ ab = ba,\ a^T = a,\ b^T = b,$$
$$F_i = \text{diag}(b^{(i)}),\ G_i = D[\bar{H}_i]^{-1},$$
$$b^{(1)} = b,\ b^{(i+1)} = b^{(i)2} - 2a^{(i)2},$$
$$a^{(1)} = a,\ a^{(i+1)} = -a^{(i)2}.$$

Sweet [S11] also uses these choices to develop an odd-even reduction algorithm. We again note that the $\bar{H}_i$ matrices lose the constant diagonal property for $i \geq 2$, but symmetry is preserved. Each block of $\bar{H}_i$ may be represented as a bivariate rational function of a and b; the scaling by F_i ensures that the blocks of block rows $j2^i$ of H_{i+1} are actually polynomials in a and b. In fact, the diagonal blocks of these rows will be $b^{(i+1)}$ and the non-zero off-diagonal blocks will be $a^{(i+1)}$.

Swarztrauber [S6] describes a Buneman-style algorithm for separable elliptic equations on a rectangle. This leads to $A = (a_j,b_j,c_j)$ where the blocks are all $n \times n$, and $a_j = \alpha_j I_n$, $b_j = T + \beta_j I_n$, $c_j = \gamma_j I_n$. In this case the blocks commute with each other and may be expressed as polynomials in the tridiagonal matrix T. Swarztrauber's odd-even elimination essentially uses $F_i = \mathrm{diag}(\mathrm{LCM}(b^{(i)}_{j-2^{i-1}}, b^{(i)}_{j+2^{i-1}}))^*$, $b^{(i)}_j = I_n$ for $j \le 0$ or $j \ge N + 1$, $G_i = D[\bar{H}_i]^{-1}$. Stone [S2] considers a similar algorithm for tridiagonal matrices ($n = 1$) but with $F_i = \mathrm{diag}(b^{(i)}_{j-2^{i-1}}\ b^{(i)}_{j+2^{i-1}})$, $G_i = D[H_i]^{-1}$. Note that symmetry is not preserved.

In all three cases it may be shown by induction that $\bar{H}_{i+1} = F_i \ldots F_1 H_{i+1}$ and $\bar{v}^{(i+1)} = F_i \ldots F_1 v^{(i+1)}$. These expressions signal the instability of the basic method when $F_j \neq I$ [B20]. However, in Section 2.A we showed that the B matrices are invariant under premultiplication by block diagonal matrices, so that $B[\bar{H}_i] = B[H_i]$.

The Buneman modifications for stability represent the $\bar{v}^{(i)}$ sequence by sequences $p^{(i)}$, $q^{(i)}$ with $\bar{v}^{(i)} = D[\bar{H}_i]p^{(i)} + q^{(i)}$. Defining $\bar{D}_i = D[\bar{H}_i]$, $\bar{Q}_i = \bar{D}_i - \bar{H}_i$, $\bar{S}_i = D[\bar{Q}_i G_i \bar{Q}_i]$ and assuming $\bar{D}_i G_i \bar{Q}_i = \bar{Q}_i G_i \bar{D}_i$, we have

$$\begin{aligned} p^{(1)} &= 0,\ q^{(1)} = \bar{v}^{(1)} = v, \\ p^{(i+1)} &= p^{(i)} + \bar{D}_i^{-1}(\bar{Q}_i p^{(i)} + q^{(i)}) \\ q^{(i+1)} &= F_i(\bar{Q}_i G_i q^{(i)} + \bar{S}_i p^{(i+1)}). \end{aligned}$$

<u>Theorem 4.3.</u> Let $\bar{P}_1 = I$, $\bar{P}_i = B[H_{i-1}] \ldots B[H_1]$, $i \ge 2$. Then $p^{(i)} = (I - \bar{P}_i)x$, $q^{(i)} = (\bar{D}_i \bar{P}_i - \bar{Q}_i)x$.

<u>Remark</u>. Although not expressed in this form, Buzbee et al. [B20] prove special cases of these formulae for the odd-even reduction applied to Poisson's equation.

<u>Proof</u>. To begin the induction, $p^{(1)} = 0 = (I - \bar{P}_1)x$, $q^{(1)} = v = \bar{H}_1 x = (\bar{D}_1 \bar{P}_1 - \bar{Q}_1)x$. Suppose $p^{(i)} = (I - \bar{P}_i)x$, $q^{(i)} = (\bar{D}_i \bar{P}_i - \bar{Q}_i)x$. Then

*LCM = least common multiple of the matrices considered as polynomials in T.

$$
\begin{aligned}
p^{(i+1)} &= (I - \bar{P}_i)x + \bar{D}_i^{-1}(\bar{Q}_i(I - \bar{P}_i)x + (\bar{D}_i\bar{P}_i - \bar{Q}_i)x) \\
&= (I - \bar{P}_i + \bar{D}_i^{-1}\bar{Q}_i - \bar{D}_i^{-1}\bar{Q}_i\bar{P}_i + \bar{D}_i^{-1}\bar{D}_i\bar{P}_i - \bar{D}_i^{-1}\bar{Q}_i)x \\
&= (I - B[\bar{H}_i]\bar{P}_i)x \\
&= (I - \bar{P}_{i+1})x
\end{aligned}
$$

and

$$
\begin{aligned}
q^{(i+1)} &= \bar{v}^{(i+1)} - \bar{D}_{i+1}p^{(i+1)} \\
&= \bar{H}_{i+1}x - \bar{D}_{i+1}(I - \bar{P}_{i+1})x \\
&= (\bar{D}_{i+1} - \bar{Q}_{i+1} - \bar{D}_{i+1} + \bar{D}_{i+1}\bar{P}_{i+1})x \\
&= (\bar{D}_{i+1}\bar{P}_{i+1} - \bar{Q}_{i+1})x. \qquad \text{QED}
\end{aligned}
$$

We note that when $G_i = D[\bar{H}_i]^{-1}$ and $\beta = \|B[A]\| < 1$ then $\|\bar{P}_i\| = \beta^{2^{i-1}-1} < 1$ and $\rho_k(\bar{D}_i\bar{P}_i - \bar{Q}_i) \le \rho_k(\bar{D}_i)\|\bar{P}_i\| + \rho_k(\bar{Q}_i) \le \rho_k(\bar{D}_i) + \rho_k(\bar{Q}_i) = \rho_k(\bar{H}_i)$, so $\|\bar{D}_i\bar{P}_i - \bar{Q}_i\| \le \|\bar{H}_i\|$. It also follows that $\|x - p^{(i)}\| \le \|\bar{P}_i\|\,\|x\|$ and $\|q^{(i)} + \bar{Q}_i x\| \le \|\bar{D}_i\bar{P}_i\|\,\|x\|$. Thus $p^{(i)}$ will be an approximation to x; Buzbee [B17] has taken advantage of this fact in a semidirect method for Poisson's equation.

4.C. Odd-Even Reduction

4.C.1. The Basic Algorithm

The groundwork for the odd-even reduction algorithms has now been laid. Returning to the original derivation of odd-even elimination in (4.1) and (4.2), suppose we collect the even-numbered equations of (4.2) into a linear system $A^{(2)}x^{(2)} = w^{(2)}$, where

$$A^{(2)} = (-a_{2j}b_{2j-1}^{-1}a_{2j-1}, b_{2j} - a_{2j}b_{2j-1}^{-1}c_{2j-1} - c_{2j}b_{2j+1}^{-1}a_{2j+1}, -c_{2j}b_{2j+1}^{-1}c_{2j+1})_{N_2},$$

$$x^{(2)} = (x_{2j})_{N_2},$$

$$w^{(2)} = (v_{2j} - a_{2j}b_{2j-1}^{-1}v_{2j-1} - c_{2j}b_{2j+1}^{-1}v_{2j+1})_{N_2} = (v_{2j}^{(2)})_{N_2}$$

$N_2 = \lfloor N/2 \rfloor$.

This is the reduction step, and it is clearly a reduction since $A^{(2)}$ is half the size of $A^{(1)} = A$. Once $A^{(2)}x^{(2)} = w^{(2)}$ has been solved for $x^{(2)}$ we can compute the remaining components of $x^{(1)} = x$ by the back substitution

$$x_{2j-1} = b_{2j-1}^{-1}(v_{2j-1} - a_{2j-1}x_{2j-2} - c_{2j-1}x_{2j}), \quad j = 1,\dots,\lceil N/2 \rceil.$$

Since $A^{(2)}$ is block tridiagonal we can apply this procedure recursively to obtain a sequence of systems $A^{(i)}x^{(i)} = w^{(i)}$, where $A^{(1)} = A$, $w^{(1)} = v$, $x^{(i)} = (x_{j2^{i}-1})_{N_i}$, $N_1 = N$, $N_{i+1} = \lfloor N_i/2 \rfloor$. We write $A^{(i)} = (a_j^{(i)}, b_j^{(i)}, c_j^{(i)})_{N_i}$, $x_j^{(i)} = x_{j2^{i}-1}$. The reduction stops with $A^{(m)}x^{(m)} = w^{(m)}$, $m = \lceil \log_2 N+1 \rceil$ (note the change in notation) since $A^{(m)}$ consists of a single block. At this point the back substitution begins, culminating in $x^{(1)} = x$.

Before proceeding, we note that if $N = 2^m-1$ then $N_i = 2^{m+1-i} - 1$. The odd-even reduction algorithms have usually been restricted to this particular choice of N, but in the general case this is not necessary. However, for $A = (a,b,a)_N$ there are a number of important simplifications that can be made if $N = 2^m-1$ [B20]. Typically these involve the special scalings mentioned earlier, which effectively avoid the inversion of the diagonal blocks during the reduction.

It is also not necessary to continue the reduction recursively if $A^{(k)}x^{(k)} = w^{(k)}$ can be solved more efficiently by some other method. This leads to various polyalgorithms and the semidirect methods of Section 7.

By the way in which $A^{(i)}$ has been constructed, it is clear that it consists of blocks taken from H_i. For example, in Figure 4.1a, $A^{(4)}$ is the (8,8) block of H_4. It follows immediately that $\|A^{(i)}\| \le \|H_i\|$, $\|B[A^{(i)}]\| \le \|B[H_i]\|$, and $\|C[A^{(i)}]\|_1 \le \|C[H^{(i)}]\|_1$. As a corollary of Theorem 4.1 we actually have

<u>Corollary 4.1.</u> If $\|B[A]\| < 1$ then $\|B[A^{(i+1)}]\| \le \|B[A^{(i)}]^2\|$ and $\|A^{(i+1)}\| \le \|A^{(i)}\|$. If $\|C[A]\|_1 < 1$ then $\|C[A^{(i+1)}]\|_1 \le \|C[A^{(i)}]^2\|_1$ and $\|A^{(i+1)}\|_1 \le \|A^{(i)}\|_1$.

Recalling the definition of Theorem 3.4, let $\lambda(A) = \max_j(4\|b_j^{-1}a_j\|\,\|b_{j-1}^{-1}c_{j-1}\|)$.

<u>Theorem 4.4.</u> If $\lambda(A) \le 1$ then, with $\lambda^{(i)} = \lambda(A^{(i)})$, we have $\lambda^{(i+1)} \le (\lambda^{(i)}/(2 - \lambda^{(i)}))^2$.

<u>Proof</u>. It will suffice to consider $i = 1$. We have

$$b_j^{(2)} = b_{2j}(1 - \sigma_{2j}),$$

$$\sigma_{2j} = b_{2j}^{-1}a_{2j}b_{2j-1}^{-1}c_{2j-1} + b_{2j}^{-1}c_{2j}b_{2j+1}^{-1}a_{2j+1}$$

$$a_j^{(2)} = -a_{2j}b_{2j-1}^{-1}a_{2j-1},$$

$$c_j^{(2)} = -c_{2j}b_{2j+1}^{-1}c_{2j+1},$$

$$\|\sigma_{2j}\| \le \|b_{2j}^{-1}a_{2j}\|\,\|b_{2j-1}^{-1}c_{2j-1}\| + \|b_{2j+1}^{-1}a_{2j+1}\|\,\|b_{2j}^{-1}c_{2j}\| \le 2(\lambda/4) = \lambda/2,$$

$$\|(1-\sigma_{2j})^{-1}\| \le (1 - \lambda/2)^{-1} = 2/(2 - \lambda),$$

$$\begin{aligned} 4\|b_j^{(2)-1}a_j^{(2)}\|\,\|b_{j-1}^{(2)-1}c_{j-1}^{(2)}\| &\le 4\|(1 - \sigma_{2j})^{-1}\|\,\|(1 - \sigma_{2j-2})^{-1}\|\,(\|b_{2j}^{-1}a_{2j}\|\,\|b_{2j-1}^{-1}c_{2j-1}\|) \\ &\quad \cdot (\|b_{2j-1}^{-1}a_{2j-1}\|\,\|b_{2j-2}^{-1}c_{2j-2}\|) \\ &\le 4(2/(2 - \lambda))^2(\lambda/4)^2 = (\lambda/(2 - \lambda))^2. \end{aligned}$$

Thus $\lambda^{(2)} \le (\lambda^{(1)}/(2 - \lambda^{(1)}))^2$. QED

This may be compared to earlier results for constant diagonal matrices, which gives an equality of the form $\beta^{(2)} = \beta^2/(2 - \beta^2)$. See also Theorem 4.8 of [V3].

4.C.2. Relation to Block Elimination

By regarding odd-even reduction as a compact form of odd-even elimination we are able to relate it to the Newton iteration for A^{-1}. It is also possible to express the reduction algorithm as block elimination on a permuted system $(PAP^T)(Px) = (Pv)$ (cf. [B11], [E3], [W2]). In Section 5 we show how this formulation also yields quadratic convergence theorems.

Specifically, let P be the permutation matrix which reorders the vector $(1,2,3,\dots,N)^T$ so that the odd multiples of 2^0 come first, followed by the odd multiples of 2^1, the odd multiples of 2^2, etc. For $N = 7$, $P(1,2,\dots,7)^T = (1,3,5,7,2,6,4)^T$, and

$$PAP^T = \begin{pmatrix} b_1 & & & & c_1 & & \\ & b_3 & & & a_3 & & c_3 \\ & & b_5 & & & c_5 & a_5 \\ & & & b_7 & & a_7 & \\ a_2 & c_2 & & & b_2 & & \\ & & a_6 & c_6 & & b_6 & \\ & a_4 & c_4 & & & & b_4 \end{pmatrix}.$$

The block factorization of $PAP^T = LU$ is

$$L = \begin{pmatrix} I & & & & & & \\ & I & & & & & \\ & & I & & & & \\ & & & I & & & \\ a_2 b_1^{-1} & c_2 b_3^{-1} & & & I & & \\ & & a_6 b_5^{-1} & c_6 b_7^{-1} & & I & \\ & a_4 b_3^{-1} & c_4 b_5^{-1} & & a_2^{(2)} (b_1^{(2)})^{-1} & c_2^{(2)} (b_3^{(2)})^{-1} & I \end{pmatrix}$$

$$U = \begin{pmatrix} b_1 & & & & c_1 & & \\ & b_3 & & & a_3 & & c_3 \\ & & b_5 & & & c_5 & a_5 \\ & & & b_7 & & a_7 & \\ & & & & b_1^{(2)} & & c_1^{(2)} \\ & & & & & b_3^{(2)} & a_3^{(2)} \\ & & & & & & b_1^{(3)} \end{pmatrix} .$$

If we let $b_j^{(i)} = \ell_j^{(i)} u_j^{(i)}$ be an LU factorization with $\ell(u)$ lower (upper) triangular then the ordinary LU factorization of $PAP^T = L'U'$ is

$$L' = \begin{pmatrix} \ell_1 & & & & & & \\ & \ell_3 & & & & & \\ & & \ell_5 & & & & \\ & & & \ell_7 & & & \\ a_2u_1^{-1} & c_2u_3^{-1} & & & \ell_1^{(2)} & & \\ & & a_6u_5^{-1} & c_6u_7^{-1} & & \ell_3^{(2)} & \\ & a_4u_3^{-1} & c_4u_5^{-1} & & a_2^{(2)}(u_1^{(2)})^{-1} & c_2^{(2)}(u_3^{(2)})^{-1} & \ell_1^{(3)} \end{pmatrix}$$

$$U' = \begin{pmatrix} u_1 & & & & \ell_1^{-1}c_1 & & \\ & u_3 & & & \ell_3^{-1}a_3 & & \ell_3^{-1}c_3 \\ & & u_5 & & & \ell_5^{-1}c_5 & \ell_5^{-1}a_5 \\ & & & u_7 & & \ell_7^{-1}a_7 & \\ & & & & u_1^{(2)} & & (\ell_1^{(2)})^{-1}c_1^{(2)} \\ & & & & & u_3^{(2)} & (\ell_3^{(2)})^{-1}a_3^{(2)} \\ & & & & & & u_1^{(3)} \end{pmatrix}.$$

This factorization gives rise to a related method, first solving $L'z = Pv$, then $U'(Px) = z$. It is easily seen that $z_j^{(i)} = (\ell_j^{(i)})^{-1}w_j^{(i)}$.

We may now apply any of the theorems about Gaussian or block elimination to PAP^T and obtain corresponding results about odd-even reduction and the above modification. In particular, Theorem 3.2 applies though its conclusions are weaker than those of Corollary 4.1. It follows from the $PAP^T = LL^7$ factorization as well as Theorem 4.2 that $A^{(i)}$ will be positive definite if A is; Theorem 4.2 also allows the conclusion that $\rho(B[A^{(i)}]) < 1$.

The odd-even reduction is also a special case of the more general cyclic reduction defined by Hageman and Varga [H1] for the iterative solution of large sparse systems. Since PAP^T is in the 2-cyclic form

$$PAP^T = \begin{pmatrix} D_1 & A_{12} \\ A_{21} & D_2 \end{pmatrix},$$

D_1 and D_2 block diagonal, we can premultiply by

$$\begin{pmatrix} I & -A_{12}D_2^{-1} \\ -A_{21}D_1^{-1} & I \end{pmatrix}$$

to obtain

$$\begin{pmatrix} D_1 - A_{12}D_2^{-1}A_{21} & 0 \\ 0 & D_2 - A_{21}D_1^{-1}A_{12} \end{pmatrix}$$

It is clear that $A^{(2)} = D_2 - A_{21}D_1^{-1}A_{12}$ and $(D_1 - A_{12}D_2^{-1}A_{21}, 0)$ are the rows of H_2 that have not been computed as part of the reduction. In consequence of this formulation, if A is an M-matrix then so $A^{(i)}$ and $\rho(B[A^{(i+1)}]) < \rho^2(B[A^{(i)}]) < 1$ [H1].

The block LU factorization $PAP^T = L\Delta U$, where Δ is block diagonal and L(U) is unit lower (upper) triangular, determines the reduction and back substitution phases. Norm bounds on L and U were given in Section 3.A.2 for general matrices with $\|B[A]\| < 1$. For block tridiagonal matrices in the natural ordering we have $\|L\|_1 \leq 1 + \|C[A]\|_1$, $\|L\| \leq n\|C[A]\|_1$ if $\|C[A]\|_1 < 1$, $\|U\| \leq 1 + \|B[A]\|$, $\|U\|_1 \leq 1 + n\|B[A]\|$ if $\|B[A]\| < 1$, $n = \max_j n_j$. For the permuted block tridiagonal system only the bounds on $\|L\|$ and $\|U\|_1$ will grow, and then only logarithmically. By the construction of L and U,

$$\|L\|_1 \le 1 + \max_{1\le i\le m} \|C[A^{(i)}]\|_1,$$

$$\|L\| \le 1 + \sum_{i=1}^{m} \|C[A^{(i)}]\|,$$

$$\|U\|_1 \le 1 + \sum_{i=1}^{m} \|B[A^{(i)}]\|_1,$$

$$\|U\| \le 1 + \max_{1\le i\le m} \|B[A^{(i)}]\|.$$

Since $C[A^{(i)}]$ is block tridiagonal, $\rho_\ell(C[A^{(i)}]) \le 2n\|C[A^{(i)}]\|_1$, so $\|C[A^{(i)}]\| \le 2n\|C[A^{(i)}]\|_1$; similarly $\|B[A^{(i)}]\|_1 \le 2n\|B[A^{(i)}]\|$. If $\|B[A]\| < 1$, $\|C[A]\|_1 < 1$, we then have

$$\begin{aligned}
\|L\|_1 &\le 1 + \|C[A]\|_1 \\
\|L\| &\le 1 + 2nm\|C[A]\|_1, \\
\|U\|_1 &\le 1 + 2nm\|B[A]\|, \\
\|U\| &\le 1 + \|N[A]\|.
\end{aligned}$$

Much like the proof of Corollary 3.4 we also have, if $\|B[A]\| < 1$,

$$b_j^{(i+1)} = b_{2j}^{(i)}(1 - \sigma_{2j}^{(i)})$$

$$\|\sigma_{2j}^{(i)}\| \le \|B[A^{(i)}]^2\|,$$

so $$\|b_j^{(i+1)}\| \le \|b_{2j}^{(i)}\|(1 + \beta_i), \quad \beta_i = \|B[A^{(i)}]^2\|,$$

$$\|(b_j^{(i+1)})^{-1}\| \le \|(b_{2j}^{(i)})^{-1}\| / (1 - \beta_i),$$

and $$\operatorname{cond}(b_j^{(i+1)}) \le \operatorname{cond}(b_{2j}^{(i)})(1 + \beta_i) / (1 - \beta_i).$$

Since our interest is in bounding $\mathrm{cond}(b_j^{(i+1)})$ in terms of $\mathrm{cond}(b_k)$, we must simplify the above expression. Let $K_i = \Pi_{j=1}^{i}(1 + \beta_j)/(1 - \beta_j)$, so that

$$\mathrm{cond}(b_j^{(i+1)}) \le [\max_k \mathrm{cond}(b_k)]K_i$$

by induction. Using $\beta_j \le \beta_{j-1}^2$, we obtain

$$\begin{aligned} K_i &\le \Pi_{j=1}^{i}(1 + \beta_j)/(1 - \beta_{j-1}^2) \\ &= (1 + \beta_i)/[(1 - \beta_1)\Pi_{j=1}^{i-1}(1 - \beta_j)] \\ &< (1 + \beta_i)/(1 - \beta_1)^i. \end{aligned}$$

This is a rather unattractive upper bound since it may be quite large if β_1 is close to 1. Actually, the situation is not so bad, since $\| \sigma_{2j}^{(i)} \| \lesssim \frac{1}{2}\| B[A^{(i)}]^2 \| = \frac{1}{2}\beta_i$, and

$$\mathrm{cond}(b_j^{(i+1)}) \lesssim [\max_k \mathrm{cond}(b_k)] \; \Pi_{j=1}^{i} \frac{2 + \beta_j}{2 - \beta_j}$$

If $\lambda(A) \le 1$ then $\| \sigma_{2j}^{(i)} \| \le \lambda^{(i)}/2$, $\lambda^{(i)} = \lambda(A^{(i)})$, and

$$\begin{aligned} \mathrm{cond}(b_j^{(i+1)}) &\le [\max_k \mathrm{cond}(b_k)] \; \Pi_{j=1}^{i} \frac{2 + \lambda^{(i)}}{2 - \lambda^{(i)}} \\ &\le 3^i \max_k \mathrm{cond}(b_k). \end{aligned}$$

When A is positive definite it follows from various eigenvalue-interlacing theorems that $\mathrm{cond}(b_j^{(i)}) < \mathrm{cond}(A^{(i)}) < \mathrm{cond}(A)$.

4.C.3. Back Substitution

Now let us consider error propagation in the back substitution. Suppose $y_k^{(i+1)} = x_k^{(i+1)} + \xi_k^{(i+1)}$, $1 \le k \le N_{i+1}$, defines the computed solution to $A^{(i+1)}x^{(i+1)} = w^{(i+1)}$, and we compute

$$y_{2j-1}^{(i)} = (b_{2j-1}^{(i)})^{-1}(w_{2j-1}^{(i)} - a_{2j-1}^{(i)}\,y_{j-1}^{(i+1)} - c_{2j-1}^{(i)}\,y_j^{(i+1)})$$
$$+ \epsilon_{2j-1}^{(i)},$$
$$y_{2j}^{(i)} = y_j^{(i+1)}.$$

Then $\xi_{2j-1}^{(i)} = y_{2j-1}^{(i)} - x_{2j-1}^{(i)}$

$$= (b_{2j-1}^{(i)})^{-1}\,a_{2j-1}^{(i)}\,\xi_{j-1}^{(i+1)} - (b_{2j-1}^{(i)})^{-1}\,c_{2j-1}^{(i)}\,\xi_j^{(i+1)}$$
$$+ \epsilon_{2j-1}^{(i)},$$
$$\xi_{2j}^{(i)} = \xi_j^{(i+1)}.$$

Now define the vectors

$$z_1 = (\epsilon_1^{(i)}, 0, \epsilon_3^{(i)}, 0, \ldots)^T$$
$$z_2 = (0, \xi_1^{(i+1)}, 0, \xi_2^{(i+1)}, \ldots)^T.$$

We have $\|z_2\| = \|\xi^{(i+1)}\|$ by construction, and

$$\xi^{(i)} = B^{(i)}z_2 + z_1 + z_2,\ B^{(i)} = B[A^{(i)}].$$

But in consequence of the construction we actually have

$$\|\xi^{(i)}\| \le \max(\|B^{(i)}z_2 + z_1\|, \|z_2\|).$$

If $\|B[A]\| < 1$ then $\|B^{(i)}z_2\| \le \|B^{(i)}\|\,\|z_2\| < \|z_2\|$.

Suppose $\|\xi^{(m)}\| \le \epsilon$ and $\|\epsilon^{(i)}\| \le \epsilon$ for all $i = 1,\ldots,m-1$. Then

$$\|\xi^{(i)}\| \le \max(\|B^{(i)}\|\,\|z_2\| + \|z_1\|, \|z_2\|)$$
$$\le \|z_2\| + \|z_1\|$$
$$= \|\xi^{(i+1)}\| + \|\epsilon^{(i)}\|$$
$$\le (m - (i+1) + 1)\epsilon + \epsilon \quad \text{(by induction on i)}$$
$$= (m - i + 1)\epsilon,$$

and $\| \xi^{(1)} \|$, the error in the computed solution of $Ax = v$, is bounded by $m\epsilon$, $m = \lceil \log_2 N + 1 \rceil$. This is certainly a satisfactory growth rate for error propagation in the back substitution. In fact, it is a somewhat pessimistic upper bound, and we can more practically expect that errors in the solution of $A^{(i+1)}x^{(i+1)} = w^{(i+1)}$ will be damped out quickly owing to the quadratic decreases in $\| B^{(i)} \|$.

4.C.4. Storage Requirements

On the matter of fill-in and storage requirements, we note that block elimination in the natural ordering of A, as discussed in Section 3.A, requires no additional block storage since there is no block fill-in. However, it has already been seen that odd-even reduction does generate fill-in. The amount of fill-in is the number of off-diagonal blocks of $A^{(2)}, \ldots, A^{(m)}$, which is $\sum_{i=2}^{m} 2(N_i - 1) < 2N$. Since the matrix A requires $3N - 2$ storage blocks, this is not an excessive increase. Of course, we have not considered the internal structure of the blocks, as this is highly dependent on the specific system, nor have we considered temporary storage for intermediate computations.

We do note, however, that if the factorization need not be saved, then some economies may be achieved. For example, in the transition from $A^{(1)}$ to $A^{(2)}$, the first equation of $A^{(2)}x^{(2)} = w^{(2)}$ is generated from the first three equations of $A^{(1)}x^{(1)} = w^{(1)}$. The first and third of these must be saved for the back substitution, but there is no reason to save the second once $A^{(2)}x^{(2)} = w^{(2)}$ is formed. Thus the first equation of $A^{(2)}x^{(2)} = w^{(2)}$ may overwrite the second equation of $A^{(1)}x^{(1)} = w^{(1)}$. Similar considerations hold throughout the reduction, but some modifications are needed on the CDC STAR-100 due to its restrictions on vector addressing (see Section 10.B).

The special scalings of odd-even reduction for systems derived from separable elliptic PDE's allow implicit representation of the blocks of $A^{(i)}$ as polynomials or rational functions of a single matrix variable. Since the blocks of A are quite sparse this is necessary in order to limit intermediate storage to a small multiple of the original storage. Unfortunately, there does not seem to be a similar procedure that can be applied to nonseparable PDE's. The only recourse is to store the blocks as dense matrices, and on current computing systems this will limit them to small sizes.

4.C.5. Special Techniques

Now suppose that $A = (-p_j, t_j + p_j + q_j, -q_j)$, $p_j, q_j, t_j \geq 0$, $p_1 = q_N = 0$; we would like to apply Babuska's tricks for the LU factorization (Section 3.A.4) to the factorization induced by odd-even reduction. This only partially succeeds. First, we can represent $A^{(2)} = (-p_j^{(2)}, t_j^{(2)} + p_j^{(2)} + q_j^{(2)}, -q_j^{(2)})$ where

$$\begin{aligned}
p_j^{(2)} &= p_{2j}\, b_{2j-1}^{-1}\, p_{2j-1},\\
t_j^{(2)} &= t_{2j} + p_{2j}\, b_{2j-1}^{-1}\, t_{2j-1} + q_{2j}\, b_{2j+1}^{-1}\, t_{2j+1},\\
q_j^{(2)} &= q_{2j}\, b_{2j+1}^{-1}\, q_{2j+1},\\
b_k &= t_k + p_k + q_k.
\end{aligned}$$

Note that $t_{2j} \leq t_j^{(2)} \leq b_{2j}$. On the other hand, the technique of avoiding the back substitution by combining the LU and UL factorizations of PAP^T will not work. Let $A' = PAP^T = (I + L)(D + U) = (I + U^*)(D^* + L^*)$, $(I + L)f = Pv$, $(D + U)(Px) = f$, $(I + U^*)f^* = Pv$, $(D^* + L^*)(Px) = f^*$, so that

$$(D + D^* - D[A'])(Px) = f + f^* - (L^* + D[A'] + U)(Px).$$

The only way to have $A' = L^* + D[A'] + U$ is if no fill-in occurs in either factorization, in which case $L^* = L[A']$, $U = U[A]$. In particular fill-in occurs in odd-even reduction, so the method does not work in its entirety. However, it will still be advantageous to do the reductions by developing the $t^{(i)}$ sequences. (See [S1] for some examples.)

Schröder and Trottenberg [S1] also consider a method of "total reduction" for matrices drawn from PDE's and finite difference approximations on a rectangular grid. Numerical stability and further developments are discussed in [S1a]. The total reduction is similar to odd-even reduction (which is derived in [S1] by a "partial reduction"), but just different enough to prevent the application of our techniques. Part of the problem seems to be that total reduction is more closely related to the PDE than to the matrix, though the algorithm is expressible in matrix form. Various convergence results are undoubtedly obtainable, but we have not yet determined the proper setting for their analysis.

4.D. Parallel Computation

Finally, the utility of the odd-even methods for parallel or pipeline computation has been demonstrated in several studies [H3]. Stone [S2], Lambiotte and Voigt [L2], and Madsen and Rodrigue [M1] discuss odd-even reduction for tridiagonal systems,* and conclude that it is the best method to use when N is large ($\gtrsim 256$). Jordan [J1] considers odd-even elimination, which is shown in [M1] to be most useful for middle ranges of N ($64 \lesssim N \lesssim 256$) on a particular pipeline computer, the CDC STAR-100. For small values of N the sequential LU factorization is best, owing to the effect of lower order terms engendered by the pipeline mechanism.

*The basic method given here is a block analogue of the method examined in [L2].

The inherent parallelism of the odd-even methods is in formation of the equations of $A^{(i+1)}x^{(i+1)} = w^{(i+1)}$ or $H_{i+1}x = v^{(i+1)}$ and in the back substitution for odd-even reduction. It is readily seen from (4.2) that H_2 and $v^{(2)}$ may be computed from H_1 and $v^{(1)}$ by

compute LU factors of b_k, $(1 \le k \le N)$

solve $b_k[\hat{a}_k \ \hat{c}_k \ \hat{v}_k] = [a_k \ c_k \ v_k]$, $(1 \le k \le N)$

$b_k^{(2)} - b_k - a_k\hat{c}_{k-1} - c_k\hat{a}_{k+1}$, $(1 \le k \le N)$

$v_k^{(2)} \leftarrow v_k - a_k\hat{v}_{k-1} - c_k\hat{v}_{k+1}$, $(1 \le k \le N)$

$a_k^{(2)} \leftarrow -a_k\hat{a}_{k-1}$, $(3 \le k \le N)$

$c_k^{(2)} \leftarrow -c_k\hat{c}_{k+1}$, $(1 \le k \le N - 2)$

(Special end conditions are handled by storing zeros strategically or by shortening vector lengths slightly. $A^{(2)}$ and $w^{(2)}$ are computed by only generating new equations for even k; the back substitution is then rather straightforward once $x^{(2)}$ is computed.

The algorithm for odd-even reduction, $N = 2^m - 1$, all blocks $n \times n$, $N_i = 2^{m+1-i} - 1$, is as follows: for $i = 1,\ldots,m-1$

compute LU factors of $b_{2k+1}^{(i)}$, $(0 \le k \le N_{i+1})$

solve $b_{2k+1}^{(i)}[\hat{a}_{2k+1} \ \hat{c}_{2k+1} \ \hat{w}_{2k+1}] = [a_{2k+1}^{(i)} \ c_{2k+1}^{(i)} \ w_{2k+1}^{(i)}]$, $(0 \le k \le N_{i+1})$

$b_k^{(i+1)} = b_{2k}^{(i)} - a_{2k}^{(i)}\ \hat{c}_{2k-1} - c_{2k}^{(i)}\ \hat{a}_{2k+1}$, $(1 \le k \le N_{i+1})$

$w_k^{(i+1)} = w_{2k}^{(i)} - a_{2k}^{(i)}\ \hat{w}_{2k-1} - c_{2k}^{(i)}\ \hat{w}_{2k+1}$, $(1 \le k \le N_{i+1})$

$a_k^{(i+1)} = -a_{2k}^{(i)}\ \hat{a}_{2k-1}$, $(2 \le k \le N_{i+1})$

$c_k^{(i+1)} = -c_{2k}^{(i)}\ \hat{c}_{2k+1}$, $(1 \le k \le N_{i+1} - 1)$

```
solve $b_1^{(m)} x_1^{(m)} = w_1^{(m)}$
for $i = m-1,\dots,1$
    solve $b_{2k+1}^{(i)} x_{2k+1}^{(i)} = (w_{2k+1}^{(i)} - a_{2k+1}^{(i)} x_k^{(i+1)} - c_{2k+1}^{(i)} x_{k+1}^{(i+1)})$, $(0 \le k \le N_{i+1})$
```

The ith reduction and corresponding back substitution uses $(12\frac{2}{3}n^3 + O(n^2))N_{i+1} - 12n^3$ arithmetic operations, and the total operation count for the algorithm is $(12\frac{2}{3}N - 12m)n^3 + O(n^2N + n^3)$. This may be compared to $4\frac{2}{3}Nn^3 + O(n^2N + n^3)$ operations for solution of $Ax = v$ by the LU factorization.

For a crude estimate of execution time on a pipeline computer,* we take $\tau N + \sigma$, $\tau \ll \sigma$, as the execution time for operations on vectors of length N, and t, $\tau \ll t \ll \sigma$, as the execution time for scalar operations [H3]. The total execution time for odd-even reduction would then be roughly $(12\frac{2}{3}n^3 + O(n^2))(\tau N + \sigma m) + O(n^3 t)$ versus $(4\frac{2}{3}n^3N + O(n^2N + n^3))t$ for the LU factorization done completely in scalar mode. A qualitative comparison shows results similar to the more precise analysis given to tridiagonal systems: for a fixed value of n, odd-even reduction will become increasingly more effective than the LU factorization as N increases. Odd-even elimination should maintain its utility for moderate values of N; the factor $\tau N + \sigma m$ is replaced by $(\tau N + \sigma)m$, but data manipulation is often much simpler than in odd-even reduction. The actual crossover points between algorithms are implementation dependent and will be considered to a limited extent in Section 10.C.

A critical observation on the execution time of odd-even reduction is that roughly 50% of the total time is spent in the first reduction from

*A more detailed timing analysis is in Section 10.C and Appendix B.

$A^{(1)} = A$ to $A^{(2)}$. Special efforts to optimize the algorithm by solving the reduced systems differently must take this fact into account. Furthermore, in timing each reduction it is seen that roughly 60% of the high-order term n^3N comes from the matrix multiplications used to compute $A^{(i+1)}$ and 30% from solving the 2n+1 linear systems for $\hat{a}_{2k+1}$, etc. Relatively little effort is used to factor the $b_j^{(i)}$'s or to make back substitutions having saved the factorizations.

The use of a synchronous parallel computer creates some constraints on implementation which are not so important on a sequential computer. The first restriction in the interests of efficiency is that the blocks of A should all be the same size. This allows simultaneous matrix operations (factoring, etc.) to be programmed without resorting to numerous special cases due to varying block sizes. In addition, pivoting causes inefficiencies when the pivot selections differ between the diagonal blocks. As with the LU factorization, the assumption $\|B[A]\| < 1$ only guarantees that no block pivoting is needed. If A is strictly diagonally dominant or positive definite, then no pivoting within the blocks will be required.

Thirdly, the choice of data structures to represent the matrices and vectors must conform to the machine's requirements for the parallel or vector operations. This is an important problem on the CDC Star owing to its limitations on vector addressing and allocation [L1]. When A is restricted to having small blocks this is not as serious as it would be otherwise. In the case of the Illiac IV, the problem of data communication between processing elements must also be considered.

4.E. Summary

We now summarize the results of this section. The odd-even elimination and reduction algorithms may be safely applied to the solution of certain block tridiagonal linear systems, and in some cases various important quantities involved in the solution decrease quadratically to zero. The quadratic convergence may be explained by expressing the methods as Newton-like iterations to compute a block diagonal matrix. Variations of the basic methods, which have important applications to PDE's, also display quadratic convergence. Parallel computation is a natural format for these methods, although practicalities place some restrictions on the type of system that can be solved efficiently.

5. UNIFICATION: ITERATION AND FILL-IN

Having presented several linear and quadratic methods, we now treat them in a unified manner as particular instances of a general nonstationary iteration. We are also lead to an interpretation of quadratic convergence in terms of matrix fill-in, and conclude that the odd-even methods are the only instances of the general iteration which display quadratic convergence. These results apply to arbitrary matrices satisfying $\|B[A]\| < 1$ and not only to the block tridiagonal case.

Let us first collect the formulae for the block LU factorization, block Gauss-Jordan, and odd-even elimination.

$$\begin{aligned}
&\text{LU:} && A_{i+1} = (I - L[A_i]E_iD[A_i]^{-1})A_i,\ A_1 = A.\\
&\text{GJ:} && \bar{A}_{i+1} = (I - (L[\bar{A}_i] + U[\bar{A}_i])E_iD[\bar{A}_i]^{-1})\bar{A}_i,\ \bar{A}_1 = A.\\
&\text{OE:} && H_{i+1} = (I - (L[H_i] + U[H_i])D[H_i]^{-1})H_i,\ H_1 = A.
\end{aligned}$$

Now consider the function $F(X,Y,Z) = (2I - XY)Z$, and observe that

$$\begin{aligned}
A_{i+1} &= F(D[A_i] + L[A_i]E_i,\ D[A_i]^{-1},\ A_i)\\
\bar{A}_{i+1} &= F(D[\bar{A}_i] + (L[\bar{A}_i] + U[\bar{A}_i])E_i,\ D[\bar{A}_i]^{-1},\ \bar{A}_i)\\
H_{i+1} &= F(H_i,\ D[H_i]^{-1},\ H_i)
\end{aligned}$$

Thus it is necessary to analyze sequences based on repeated applications of F.

For fixed nonsingular Y we can define the residual matrices $r(M) = I - MY$, $s(M) = I - YM$, so $F(X,Y,Z) = (I + r(X))Z$ and

$$\begin{aligned}
r(F(X,Y,Z)) &= r(Z) - r(X) + r(X)r(Z),\\
s(F(X,Y,Z)) &= s(Z) - s(X) + s(X)s(Z).
\end{aligned}$$

By taking $X = Z$ and assuming that $\| r(Z_1) \| < 1$ or $\| s(Z_1) \| < 1$, the stationary iteration $Z_{i+1} = F(Z_i,Y,Z_i)$ converges quadratically to Y^{-1}. This is, of course, the Newton iteration.

The related operator $G(X,Y,Z) = Z + X(I - YZ) = Z + Xs(Z)$ has previously been studied as a means of solving linear systems [H8]. When $\bar{Z}_1 = X$, $\bar{Z}_{i+1} = G(X,Y,\bar{Z}_i)$, Bauer [B8] observes that $r(\bar{Z}_{i+1}) = r(X)r(\bar{Z}_i)$, $s(\bar{Z}_{i+1}) = s(X)s(\bar{Z}_i)$, $\bar{Z}_i = (I - r(X)^i)Y^{-1} = Y^{-1}(I - s(X)^i)$, $\bar{Z}_{i+k} = G(\bar{Z}_k,Y,\bar{Z}_i)$, and in particular $\bar{Z}_{2i} = G(\bar{Z}_i,Y,\bar{Z}_i)$. This last formula also leads to the Newton iteration, for $G(Z,Y,Z) = F(Z,Y,Z)$, $s(\bar{Z}_{2i}) = s(\bar{Z}_i)^2$.

In this section we study the nonstationary iteration $Z_1 = A$, $Z_{i+1} = F(X_i,Y_i,Z_i)$, where $Y_i = D[Z_i]^{-1}$ and X_i consists of blocks taken from Z_i, including the diagonal. The selection of blocks for X_i is allowed to change from step to step, but we shall assume that the selection sequence is predetermined and will not be altered adaptively. (The cost of such an adaptive procedure, even on a highly parallel computer, would be prohibitive.)

For the solution of a linear system $Ax = v$ we take $(Z_1,v_1) = (A,v)$, $(Z_{i+1}, v_{i+1}) = (2I - X_iY_i)(Z_i,v_i)$, with the sequence ending in or approaching (Z^*,v^*), where Z^* is block diagonal. Then $x = Z^{*-1}v^*$. One necessary condition for consistency is that $2I - X_iY_i = I + r_i(X_i)$ be nonsingular, which holds if and only if $\rho(r_i(X_i)) < 1$. Since $r_i(X_i)$ and $s_i(X_i) = I - Y_iX_i$ are similar we could also assume $\rho(s_i(X_i)) < 1$. Using earlier notation, $r_i(X_i) = C[X_i]$, $s_i(X_i) = B[X_i]$; the notation has been changed to provide greater generality. In the context of an iteration converging to a block diagonal matrix, the condition $\| B[A]\| < 1$ (implying $\| s_1(X_1) \| \le \|s_1(Z_1) \| < 1$) states that A is sufficiently close to a particular block diagonal matrix, namely $D[A]$.

We first generalize earlier results on linear methods ($\| s_i(Z_i) \| < 1 \Rightarrow \| s_{i+1}(Z_{i+1}) \| \le \| s_i(Z_i) \|$) and then determine conditions under which we obtain a quadratic method ($\| s_i(Z_i) \| < 1 \Rightarrow \| s_{i+1}(Z_{i+1}) \| \le \| s_i(Z_i) \|^2$).

Theorem 5.1 In the sequence $Z_{i+1} = F(X_i, Y_i, Z_i)$, suppose that

$$\{(p,p) \mid 1 \le p \le N\} \subset T_i \subset \{(p,q) \mid 1 \le p,q \le N\} = U,$$
$$X_i = \Sigma_{T_i} E_p Z_i E_q, \quad Y_i = D[Z_i]^{-1},$$
$$s_i(M) = I - Y_i M.$$

Then $\| s_1(Z_1) \| < 1$ implies

$$\| s_{i+1}(Z_{i+1}) \| \le \| s_i(Z_{i+1}) \| \le \| s_i(Z_i) \| .$$

Proof. Suppose $\| s_i(Z_i) \| < 1$. Define Q by $Z_{i+1} = Z_i - D[Z_i]Q$, so $Q = -Y_i Z_i + Y_i X_i Y_i Z_i = -s_i(X_i) + s_i(X_i) s_i(Z_i)$. Since $D[s_i(X_i)] = 0$, $S = D[Q] = D[s_i(X_i) s_i(Z_i)]$. Since $s_i(X_i) = \Sigma_{T_i} E_p s_i(Z_i) E_q$, $\| s_i(X_i) \| \le \| s_i(Z_i) \|$ and $\| S \| \le \| s_i(Z_i) \|^2 < 1$. Thus $D[Z_{i+1}] = D[Z_i](I - S)$, $Y_{i+1} = (I - S)^{-1} Y_i$, and $s_i(Z_{i+1}) = (I - S) s_{i+1}(Z_{i+1}) + S$. Lemma 2.2 applies, so that $\| s_i(Z_{i+1}) \| < 1$ implies $\| s_{i+1}(Z_{i+1}) \| \le \| s_i(Z_{i+1}) \|$. But $s_i(Z_{i+1}) = S_i(Z_i) + Q =$ $s_i(Z_i) - s_i(X_i) + s_i(X_i) s_i(Z_i) = \Sigma_{T_i'} E_p s_i(Z_i) E_q + \Sigma_{T_i} E_p s_i(Z_i) E_q s_i(Z_i)$, $T_i' = U - T_i$, so

$$\begin{aligned}\rho_j(s_i(Z_{i+1})) &\le \rho_j(\Sigma_{T_i'} E_p s_i(Z_i) E_q) \\ &\quad + \rho_j(\Sigma_{T_i} E_p s_i(Z_i) E_q) \| s_i(Z_i) \| \\ &\le \rho_j(s_i(Z_i)).\end{aligned}$$

Thus $\| s_i(Z_{i+1}) \| \le \| s_i(Z_i) \| < 1$. QED.

To see when the method is actually quadratic, we first establish that at least some blocks of $s_{i+1}(Z_{i+1})$ are bounded in norm by $\| s_i(Z_i) \|^2$.

Corollary 5.1. With the hypotheses of Theorem 5.1,

$$\Sigma_{T_i} E_p s_{i+1}(Z_{i+1}) E_q \| \le \| s_i(Z_i) \|^2 .$$

Proof. From

$$(5.1) \quad (I - S)\, s_{i+1}(Z_{i+1}) + S = s_i(Z_i) - s_i(X_i) + s_i(X_i) s_i(Z_i),$$
$$S = D[s_i(X_i) s_i(Z_i)],$$

and $E_p(I - S) = (I - S)E_p$, we obtain

$$\begin{aligned}(I - S)\, \Sigma_{T_i} E_p\, s_{i+1}(Z_{i+1}) E_q + S \\ = \Sigma_{T_i} E_p\, s_i(Z_i) E_q - \Sigma_{T_i} E_p\, s_i(X_i) E_q \\ + \Sigma_{T_i} E_p\, s_i(X_i) s_i(Z_i) E_q .\end{aligned}$$

But $\Sigma_{T_i} E_p\, s_i(Z_i) E_q = s_i(X_i) = \Sigma_{T_i} E_p\, s_i(X_i) E_q$,
so $(I - S)\, \Sigma_{T_i} E_p\, s_{i+1}(Z_{i+1}) E_q = \Sigma_{T_i} E_p\, s_i(X_i)\, s_i(Z_i) E_q - S$. By Lemma 2.1,
$\| \Sigma_{T_i} E_p\, s_{i+1}(Z_{i+1}) E_q \| \le \| \Sigma_{T_i} E_p\, s_i(X_i) s_i(Z_i) E_q \| \le \| s_i(X_i) s_i(Z_i) \| \le \| s_i(Z_i) \|^2$.

QED

In certain cases Corollary 5.1 provides no useful information, since we could have $\Sigma_{T_i} E_p\, s_{i+1}(Z_{i+1}) E_q = 0$. Block elimination is one example where this occurs. However, it is seen that if we are to have $\| s_{i+1}(Z_{i+1}) \| \le \| s_i(Z_i) \|^2$ then we must have

$$(5.2) \quad \| \Sigma_{T_i'} E_p\, s_{i+1}(Z_{i+1}) E_q \| \le \| s_i(Z_i) \|^2 .$$

This is certainly true if T_i' is the null set, in which case we are generating the H_i sequence, so we shall assume that T_i' is non-empty and show that (5.2) will not hold in general.

We set one final condition on T_i for notational purposes, though it turns out to have practical consequences: if it is true that $E_p Z_i E_q = 0$ for any starting matrix Z_1, then (p,q) must be in T_i. For example, we now write the Gauss-Jordan algorithm as

$$\bar{A}_{i+1} = F(D[\bar{A}_i] + (L[\bar{A}_i] + U[\bar{A}_i])(\Sigma_{j=1}^{i} E_j),\ D[\bar{A}_i]^{-1},\ \bar{A}_i).$$

Inclusion in X_i of all blocks in Z_i known to be zero clearly will not affect the sequence $Z_{i+1} = F(X_i, Y_i, Z_i)$. With the additional condition on T_i, it follows that if $(p,q) \notin T_i$ then $E_p Z_i E_q$ can be zero only by accident, and not by design.

Corollary 5.2. With the hypotheses of Theorem 5.1, only the choice $T_i' = \emptyset$ yields a quadratic method for arbitrary starting matrices Z_1.

Proof. Suppose that $T_i' \neq \emptyset$. Following the proof of Corollary 5.1, we obtain

$$\begin{aligned}(I - S)\ &\Sigma_{T_i'} E_p\ s_{i+1}(Z_{i+1}) E_q \\ &= \Sigma_{T_i'} E_p\ s_i(Z_i) E_q + \Sigma_{T_i'} E_p\ s_i(X_i) s_i(Z_i) E_q\end{aligned}$$

which implies, by Lemma 2.1,

$$\begin{aligned}&\| \Sigma_{T_i'} E_p\ s_{i+1}(Z_{i+1}) E_q \| \\ &\quad \leq \| \Sigma_{T_i'} E_p\ s_i(Z_i) E_q + S + \Sigma_{T_i'} E_p\ s_i(X_i) s_i(Z_i) E_p \| .\end{aligned}$$

The presence of nonzero 1st order terms $(\Sigma_{T_i'} E_p s_i(Z_i) E_q)$ in general dominates the 2nd order terms $(S + \Sigma_{T_i'} E_p\ s_i(X_i) s_i(Z_i) E_q)$, and we cannot conclude that (5.2) will hold for arbitrary Z_1 satisfying $\| s_1(Z_1) \| < 1$. QED

Now let us consider what happens when block fill-in occurs, as it will during the odd-even algorithms for block tridiagonals. For the off-diagonal (p,q) block, $p \neq q$, we have $E_p Z_i E_q = 0$ but $E_p Z_{i+1} E_q \neq 0$. Since $Z_{i+1} = 2Z_i - X_i Y_i Z_i$, $E_p Z_{i+1} E_q = E_p X_i (-Y_i Z_i) E_q = E_p X_i \; s_i(Z_i) E_q$, and $\| E_p Z_{i+1} E_q \| \leq \| E_p X_i \| \; \| s_i(Z_i) E_q \|$. From (5.1) and $E_p S E_q = 0$, we have $(I - S) E_p \; s_{i+1}(Z_{i+1}) E_q = E_p \; s_i(X_i) \; s_i(Z_i) E_q$, which implies $\| E_p \; s_{i+1}(Z_{i+1}) E_q \| \leq \| S + E_p \; s_i(X_i) \; s_i(Z_i) E_q \| \leq \| s_i(Z_i) \|^2$. (We cannot conclude that $\| E_p \; s_{i+1}(Z_{i+1}) E_q \| \leq \| s_{i+1}(Z_{i+1}) \|^2$, however.)

As shown by these inequalities, fill-in produces small off-diagonal blocks of Z_{i+1} when $\| s_i(Z_i) \| < 1$, and the corresponding blocks of $s_{i+1}(Z_{i+1})$ are quadratically small with respect to $s_i(Z_i)$. Thus one source of quadratic convergence is from block fill-in during the process $Z_{i+1} = F(X_i, Y_i, Z_i)$.

In summary, we now have two explanations for quadratic convergence in the odd-even methods when applied to block tridiagonal matrices. The first is that the method is a variation of the quadratic Newton iteration for A^{-1}. The second is that, in going from H_i to H_{i+1}, all the non-zero off-diagonal blocks of H_i are eliminated and all the non-zero off-diagonal blocks of H_{i+1} arise from block fill-in. Each block of $B[H_{i+1}]$ is therefore either zero or quadratic in $B[H_i]$.

6. HIGHER-ORDER METHODS

For Poisson's equation, where $A = (-I, b, -I)$, the choice $N = 2^m - 1$ leads to considerable simplification in the Buneman odd-even reduction algorithm. The most important properties are $A^{(i)} = (-I, b^{(i)}, -I)$ (constant block diagonals) and the expression of $b^{(i)}$ as a polynomial in b. However, in some applications other choices of N are much more natural. Sweet [S10] therefore described a generalization of odd-even reduction which could handle these cases and still preserve constant block diagonals and a polynomial representation. Bank's generalized marching algorithm [B5] for Poisson's equation uses one step of (very nearly the same) generalized reduction in order to control the stability of a fast marching algorithm. This technique is analagous to multiple shooting methods for ordinary differential equations.

In this section we analyze Sweet's method as yet another variation of odd-even reduction, and show that it possesses higher-order convergence properties. The algorithm actually considered here yields Sweet's method through a suitable block diagonal scaling, and hence our results carry through much as before. It must be emphasized, though, that the methods of this section are not intended as practical methods for general block tridiagonal systems. Rather, our purpose is to elaborate on the quadratic properties of odd-even reduction by showing that they are special cases of convergence properties for a family of methods.

6.A. p-Fold Reduction

The idea behind the generalization of odd-even reduction is to take a block tridiagonal linear system involving the unknowns $x_1, x_2, \ldots,$ and to produce a new, smaller block tridiagonal system involving only the unknowns $x_p, x_{2p}, \ldots$. Once this system is solved, the knowns $x_p, \ldots$ are back-substituted into the original equations to compute the remaining unknowns. The entire process will be called p-fold reduction; odd-even reduction is the case $p = 2$.

Let $p \geq 2$ be a given integer, and define $N_2 = \lfloor N/p \rfloor$. Starting with $Ax = v$ we generate the reduced system $A^{(2)}x^{(2)} = w^{(2)}$, where $A^{(2)} = (a_j^{(2)}, b_j^{(2)}, c_j^{(2)})_{N_2}$, $x^{(2)} = (x_{pj})_{N_2}$, and the diagonal block $b_j^{(2)}$ has dimension n_{pj}. For $1 \leq k \leq N_2 + 1$, define

$$\Delta_k = \begin{pmatrix} b_{kp-p+1} & c_{kp-p+1} & & \\ a_{kp-p+2} & b_{kp-p+2} & c_{kp-p+2} & \\ & \ddots & \ddots & \ddots \\ & & a_{kp-1} & b_{kp-1} \end{pmatrix}.$$

When $kp-1 > N$, either extend $Ax = v$ with trivial equations or let Δ_k be redefined as a smaller matrix using the available blocks of A. Next, repartition A to have diagonal blocks

$$\Delta_1, b_p, \Delta_2, b_{2p}, \ldots, b_{N_2p}, \Delta_{N_2+1},$$

and call the new system $\tilde{A}\tilde{x} = \tilde{v}$. Three adjacent block rows of $\tilde{A}$ are thus

$$\begin{pmatrix} \begin{pmatrix} a_{kp-p+1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} & \begin{pmatrix} & & \\ & \Delta_k & \\ & & \end{pmatrix} & \begin{pmatrix} 0 \\ \vdots \\ 0 \\ c_{kp-1} \end{pmatrix} & & \\ & (0 \ \ldots \ 0 \ a_{kp}) & (\ b_{kp} \ ) & (c_{kp} 0 \ldots 0) & \\ & & \begin{pmatrix} a_{kp+1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} & \begin{pmatrix} & & \\ & \Delta_{k+1} & \\ & & \end{pmatrix} & \begin{pmatrix} 0 \\ \vdots \\ 0 \\ c_{kp+p-1} \end{pmatrix} \end{pmatrix}$$

The p-fold reduction is obtained by performing a 2-fold reduction on $\tilde{A}$, and the back substitution is obtained by solving the system

$$\Delta_k \begin{pmatrix} x_{kp-p+1} \\ \vdots \\ x_{kp-1} \end{pmatrix} = \begin{pmatrix} v_{kp-p+1} \\ \vdots \\ v_{kp-1} \end{pmatrix} - \begin{pmatrix} a_{kp-p+1} \ x_{kp-p} \\ 0 \\ \vdots \\ 0 \\ c_{kp-1} \ x_{kp} \end{pmatrix} .$$

By taking

$$\begin{pmatrix} \alpha_{kp-p+1} & \gamma_{kp-p+1} & \varphi_{kp-p+1} \\ \vdots & \vdots & \vdots \\ \alpha_{kp-1} & \gamma_{kp-1} & \varphi_{kp-1} \end{pmatrix} = \Delta_k^{-1} \begin{pmatrix} a_{kp-p+1} & 0 & v_{kp-p+1} \\ 0 & \vdots & \vdots \\ \vdots & 0 & \vdots \\ 0 & c_{kp-1} & v_{kp-1} \end{pmatrix}$$

the reduction step simplies to

$$\begin{aligned} a_k^{(2)} &= -a_{kp} \ \alpha_{kp-1} \\ b_k^{(2)} &= b_{kp} - a_{kp} \ \gamma_{kp-1} - c_{kp} \ \alpha_{kp+1} \\ c_k^{(2)} &= -c_{kp} \ \gamma_{kp+1} \\ w_k^{(2)} &= v_{kp} - a_{kp} \ \varphi_{kp-1} - c_{kp} \ \varphi_{kp+1}, \end{aligned} \tag{6.1}$$

and the back substitution becomes

$$x_{kp-j} = \varphi_{kp-j} - \alpha_{kp-j}\, x_{kp-p} - \gamma_{kp-j}\, x_{kp},$$
$$1 \le j \le p-1.$$

Rather than computing the entire α and γ vectors, a more efficient technique is to use LU and UL processes on the Δ blocks:

```
for each k = 1,2,...,$N_2$ + 1,
    (LU process on $\Delta_k$)
    $\alpha_{k,1} = a_{kp-p+1}$; $\beta_{k,1} = b_{kp-p+1}$;
    $\varphi_{k,1} = v_{kp-p+1}$;
    for j = 2,...,p-1
        solve $\beta_{k,j-1}\,[A_{k,j-1}\; C_{k,j-1}\; F_{k,j-1}]$
                    $= [\alpha_{k,j-1}\; c_{kp-p+j-1}\; \varphi_{k,j-1}]$
        $\alpha_{k,j} = \qquad - a_{kp-p+j}\, A_{k,j-1}$
        $\beta_{k,j} = b_{kp-p+j} - a_{kp-p+j}\, C_{k,j-1}$
        $\varphi_{k,j} = v_{kp-p+j} - a_{kp-p+j}\, F_{k,j-1}$
    solve $\beta_{k,p-1}[\alpha_{kp-1}\; \gamma_{kp-1}\; \varphi_{kp-1}]$
                $= [\alpha_{k,p-1}\; c_{kp-1}\; \varphi_{k,p-1}]$
```

```
for each $k = 1,2,\dots,N_2$,
    (UL process on $\Delta_{k+1}$)
    $\beta_{k+1,-1} = b_{kp+p-1}$; $\gamma_{k+1,-1} = c_{kp+p-1}$;
    $\varphi_{k+1,-1} = v_{kp+p-1}$
    for $j = 2,\dots,p-1$
        solve $\beta_{k+1-j+1} [A_{k+1,-j+1}\ C_{k+1,-j+1}\ F_{k+1,-j+1}]$
              $= [a_{kp+p-j+1}\ \gamma_{k+1,-j+1}\ \varphi_{k+1,-j+1}]$
        $\beta_{k+1,-j} = b_{kp+p-j} - c_{kp+p-j} A_{k+1,-j+1}$
        $\gamma_{k+1,-j} = \qquad - c_{kp+p-j} C_{k+1,-j+1}$
        $\varphi_{k+1,-j} = v_{kp+p-j} - c_{kp+p-j} F_{k+1,-j+1}$
    solve $\beta_{k+1,-p+1} [\alpha_{kp+1}\ \gamma_{kp+1}\ \varphi_{kp+1}]$
          $= [a_{kp+1}\ \gamma_{k+1,-p+1}\ \varphi_{k+1,-p+1}]$
```

The block tridiagonal system $A^{(2)}x^{(2)} = w^{(2)}$ is now generated by (6.1) and solved by some procedure, and we take advantage of the earlier LU process for Δ_k in the back substitution:

```
for each $k = 1,2,\dots,N_2 + 1$,
    $x_{kp-1} = \varphi_{kp-1} - \alpha_{kp-1} x_{kp-p} - \gamma_{kp-1} x_{kp}$
    for $j = p-2,\dots,1$
        $x_{kp-p+j} = F_{k,j} - A_{k,j} x_{kp-p} - C_{k,j} x_{kp-p+j+1}$.
```

6.B. Convergence Rates

As observed in Section 2, if $\| B[A] \| < 1$ then, under the $\bar{A}$ partitioning for p-fold reduction, we have $\| B[\bar{A}] \| \le \| B[A] \|$. By Corollary 4.1 it follows that p-fold reduction will yield $\| A^{(2)} \| \le \| \bar{A} \| = \| A \|$ and $\| B[A^{(2)}] \| \le \| B[\bar{A}]^2 \| \le \| B[A] \|^2$. In fact, we can provide pth order convergence, although our first result is somewhat weaker than might be expected in comparison to earlier results.

We will actually consider a natural generalization of odd-even elimination. Let M_p be a nonsingular block (2p-1)-diagonal matrix,

$$M_p = (m_{j,-p+1},\dots,m_{j,0},\dots,m_{j,p-1})_N.$$

We want to pick M_p such that

$$M_pA = (r_j, \underbrace{0, \dots, 0}_{p-1}, s_j, \underbrace{0, \dots, 0}_{p-1}, t_j).$$

M_pA is a block (2p+1)-diagonal matrix; $A^{(2)}$ is constructed by selecting the intersections of the block rows p,2p,..., and block columns p,2p,... . Thus we have

$$a_j^{(2)} = r_{jp},\ b_j^{(2)} = s_{jp}$$
$$c_j^{(2)} = t_{jp},\ w_j^{(2)} = (M_pv)_{jp}.$$

Now, the jth block row of M_pA is

$$\sum_{k=-p+1}^{p-1} m_{j,k}[\text{block row } (j+k) \text{ of } A],$$

so that

$$r_j = m_{j,-p+1}\, a_{j-p+1},$$
$$s_j = m_{j,-1}\, c_{j-1} + m_{j,0}\, b_j + m_{j,+1}\, a_{j+1}$$
$$t_j = m_{j,p-1}\, c_{j+p-1}.$$

We require that

$$m_{j,k-1}\, c_{j+k-1} + m_{j,k}\, b_{j+k} + m_{j,k+1}\, a_{j+k+1} = 0,$$
$$|k| = 1,2,\dots,p-1,$$

where we take $m_{j,-p} = 0$, $m_{j,p} = 0$.

The special case $p = 2$, $M_p = (-a_j b_{j-1}^{-1}, I, -c_j b_{j+1}^{-1})$, has been considered in Section 4.A. For $p > 2$, define

$$
\begin{aligned}
d_{j,-p+1} &= b_{j-p+1} \\
d_{j,-k} &= b_{j-k} - a_{j-k}\, d_{j,-k-1}^{-1}\, c_{j-k-1}, \\
&\qquad k = p-2, \ldots, 1 \\
\mu_{j,-k} &= -a_{j-k}\, d_{j,-k-1}^{-1}, \\
&\qquad k = p-2, \ldots, 0 \\
d_{j,p-1} &= b_{j+p-1} \\
d_{j,k} &= b_{j+k} - c_{j+k}\, d_{j,k+1}^{-1}\, a_{j+k+1}, \\
&\qquad k = p-2, \ldots, 1, \\
\nu_{j,k} &= -c_{j+k}\, d_{j,k+1}^{-1}, \\
&\qquad k = p-2, \ldots, 0.
\end{aligned}
$$

Then we may take

$$
\begin{aligned}
m_{j,-k} &= \mu_{j,0}\, \mu_{j,-1} \cdots \mu_{j,-k+1} \\
m_{j,0} &= I \\
m_{j,k} &= \nu_{j,0}\, \nu_{j,1} \cdots \nu_{j,k-1}, \\
&\qquad k = 1, 2, \ldots, p-1.
\end{aligned}
$$

The $d_{j,-k}$'s are derived by an LU factorization process, and the $d_{j,+k}$'s by a UL process. It follows from earlier results that if $\| C[A] \|_1 < 1$ then $\| m_{j,\pm k} \|_1 \le \| C[A] \|_1^k$, so the diagonals of M_p decrease in size as one moves away from the main diagonal.

Our first theorem partially generalizes Theorem 4.1.

Theorem 6.1. Let $J = (-s_j^{-1}r_j, 0, \ldots, 0, -s_j^{-1}t_j)$, $B = (-b_j^{-1}a_j, 0, -b_j^{-1}c_j)$, $B = L + U$, $J = J_L + J_U$, $\|L\| \le \alpha$, $\|U\| \le \alpha$, $\|B\| \le 2\alpha = \beta$, with $\alpha \le \frac{1}{2}$. Then $\|J\| \le \beta^p$, $\|J_L\| \le \frac{1}{2}\beta^p$, $\|J_U\| \le \frac{1}{2}\beta^p$.

Proof. Let $R_0 = 0$, $R_k = L(I - R_{k-1})^{-1}U$, $T_0 = 0$, $T_k = U(I - T_{k-1})^{-1}L$, $k = 1,\ldots,p-1$, $S = R_{p-1} + T_{p-1}$, so that

$$(I - S)J_L = L(I - R_{p-2})^{-1}L(I - R_{p-3})^{-1}L \ldots L(I - R_0)^{-1}L,$$
$$(I - S)J_U = U(I - T_{p-2})^{-1}U(I - T_{p-3})^{-1}U \ldots U(I - T_0)^{-1}U.$$

Define $\psi_0 = 0$, $\psi_k = \alpha^2(1 - \psi_{k-1})^{-1}$, so

$$\|R_k\| \le \psi_k,\quad \|T_k\| \le \psi_k,$$
$$\|J_L\|,\ \|J_U\| \le \alpha^p/[(1 - 2\psi_{p-1})\Pi_{k=0}^{p-2}(1 - \psi_k)],$$
$$\|J\| \le \beta^p/[2^{p-1}(1 - 2\psi_{p-1})\Pi_{k=0}^{p-2}(1 - \psi_k)].$$

Define $\delta_0 = 0$, $\delta_1 = 1$, $\delta_{k+1} = \delta_k - \alpha^2\delta_{k-1}$, so $\delta_k = \det(\alpha,1,\alpha)_{(k-1)\times(k-1)} > 0$, $\psi_k = \alpha^2\delta_k/\delta_{k+1}$, $1 - \psi_k = \delta_{k+2}/\delta_{k+1}$, $(1 - 2\psi_{p-1})\Pi_{k=0}^{p-2}(1 - \psi_k) = (1 - 2\psi_{p-1})\delta_p = \delta_{p+1} - \alpha^2\delta_{p-1}$. Define $\tau_k = \delta_{k+1} - \alpha^2\delta_{k-1}$, so $\tau_1 = 1$, $\tau_2 = 1 - 2\alpha^2$, $\tau_k = \tau_{k-1} - \alpha^2\tau_{k-2}$. By induction, $\tau_k(\frac{1}{2}) = 2^{-k+1}$, $d\tau_k/d\alpha = -2\alpha k\delta_{k-1} \le 0$. But then, for $\alpha \le \frac{1}{2}$, $\tau_k(\alpha) \ge \tau_k(\frac{1}{2})$, so $2^{p-1}\tau_p(\alpha) \ge 1$. Thus $\|J\| \le \beta^p$ and $\|J_L\|$, $\|J_U\| \le \frac{1}{2}\beta^p$. QED.

Corollary 6.1. If $\|B[A]\| < 1$, $\|L[B[A]]\|$, $\|U[B[A]]\| \le \frac{1}{2}\|B[A]\|$, then $\|B[A^{(2)}]\| \le \|B[A]\|^p$.

In generalization of Theorem 4.4, there is the very surprising

Theorem 6.2. With $\lambda^{(i)} = \lambda(A^{(i)})$, $\lambda^{(1)} \le 1$, $\mu^{(i)} = (1-\sqrt{1-\lambda^{(i)}})/(1+\sqrt{1-\lambda^{(i)}})$, we have

$$\lambda^{(i+1)} \le \lambda^{(i)p}\left(\frac{1 + \mu^{(i)}}{2}\right)^{2p}\left(\frac{2}{1 + \mu^{(i)p}}\right)^{2}.$$

<u>Proof</u>. It suffices to consider $i = 1$, so we will drop the superscripts. Define $\varepsilon_{j,-k} = b^{-1}_{j-k}\, d_{j,-k}$, $\eta_{j,k} = b^{-1}_{j+k}\, d_{j,k}$, $k = 0,\dots,p-1$, so that

$$\begin{aligned}
r_j &= (-a_j\, \varepsilon^{-1}_{j,-1})(-b^{-1}_{j-1}\, a_{j-1}\, \varepsilon^{-1}_{j,-2}) \cdots (-b^{-1}_{j-p+2}\, a_{j-p+2}\, \varepsilon^{-1}_{j,-p+1}) \\
&\quad \times (b^{-1}_{j-p+1}\, a_{j-p+1}), \\
a^{(2)}_j &= r_{jp}, \\
t_j &= (-c_j\, \eta^{-1}_{j,1})(-b^{-1}_{j+1}\, c_{j+1}\, \eta^{-1}_{j,2}) \cdots (-b^{-1}_{j+p-2}\, c_{j+p-2}\, \eta^{-1}_{j,p-1}) \\
&\quad \times (b^{-1}_{j+p-1}\, a_{j+p-1}), \\
c^{(2)}_j &= t_{jp}, \\
b^{(2)}_j &= s_{jp}, \\
s_j &= b_j - a_j\, \varepsilon^{-1}_{j,-1}\, b^{-1}_{j-1}\, c_{j-1} - c_j\, \eta^{-1}_{j,1}\, b^{-1}_{j+1}\, a_{j+1} \\
&= b_j(I - \sigma_j).
\end{aligned}$$

Now, $\|\eta^{-1}_{j,p-1}\| = 1$, $\|\eta^{-1}_{j,k}\| \le (1 - \lambda\|\eta^{-1}_{j,k+1}\|/4)^{-1}$. By defining $E_1 = 1$, $E_k = (1 - \lambda E_{k-1}/4)^{-1}$, we have $\|\eta^{-1}_{j,p-k}\| \le E_k$. Similarly, $\|\varepsilon^{-1}_{j,-p+k}\| \le E_k$. This yields $\|\sigma_j\| \le \lambda E_{p-1}/2$, $\|(I - \sigma_j)^{-1}\| \le 2/(2 - \lambda E_{p-1})$. To bound $\lambda^{(2)}$, observe that

$$\begin{aligned}
&4\|b^{(2)-1}_j a^{(2)}_j\|\,\|b^{(2)-1}_{j-1} c^{(2)}_{j-1}\| \\
&\le 4\|(I - \sigma_{jp})^{-1}\|\,\|b^{-1}_{jp}\, a_{jp}\|\,\|\varepsilon^{-1}_{jp,-1}\|\,\|b^{-1}_{jp-1}\, a_{jp-1}\| \\
&\quad \times \|\varepsilon^{-1}_{jp,-2}\| \cdots \|\varepsilon^{-1}_{jp,-p+1}\|\,\|b^{-1}_{jp-p+1}\, a_{jp-p+1}\| \\
&\quad \times \|(1 - \sigma_{jp-p})^{-1}\|\,\|b^{-1}_{jp-p}\, c_{jp-p}\|\,\|\eta^{-1}_{jp-p,1}\|\,\|b^{-1}_{jp-p+1}\, c_{jp-p+1}\| \\
&\quad \times \|\eta^{-1}_{jp-p,2}\| \cdots \|\eta^{-1}_{jp-p,p-1}\|\,\|b^{-1}_{jp-1}\, c_{jp-1}\| \\
&\le 4\left(\frac{2}{2 - \lambda E_{p-1}}\right)^{2} \left(\frac{\lambda}{4}\right)^{p} \Pi^{p-1}_{j=1} E^2_j.
\end{aligned}$$

Thus $\lambda^{(2)}$ is bounded by the above quantity. Suppose first that $\lambda = 1$, which implies that $\mu = 1$. Then $E_n = 2n/(n+1)$, the bound simplifies to λ^p, and we are done with this case. Now suppose that $\lambda < 1$. It may be shown by induction that $E_n = (1 + \mu)(1 - \mu^n)/(1 - \mu^{n+1})$, yielding $(2/(2 - \lambda E_{p-1}))^2 = (1 - \lambda)^{-1}((1 - \mu^p)/(1 + \mu^p))^2$, and with a little effort the bound simplifies to the desired result. QED

Note that we have not actually shown that $\lambda^{(i)} \leq 1$ implies $\lambda^{(i+1)} \leq 1$. This point is now to be resolved.

Corollary 6.2. If $\lambda^{(i)} \leq 1$ then $\lambda^{(i+1)} \leq \lambda^{(i)p}$ with strict inequality if $\lambda^{(i)} < 1$, and $\mu^{(i+1)} \leq \mu^{(i)p}$.

Proof. The case $\lambda^{(i)} = 1$ was considered in Theorem 6.2. Suppose that $0 \leq \lambda < 1$, again considering only the case $i = 1$. It is not hard to show that $((1 + \mu)/2)^p(2/(1 + \mu^p)) = (\Pi_p(\lambda))^{-1}$, where $\Pi_p(\lambda) = \sum_{i=0}^{\lfloor p/2 \rfloor} \binom{p}{2i}(1 - \lambda)^i$. Now, $\Pi_p(1) = 1$, $\Pi'_p(\lambda) < 0$ for $0 \leq \lambda < 1$, so $\Pi_p(\lambda) > 1$ for $0 \leq \lambda < 1$ and $\lambda^{(2)} < \lambda^p$ for this case. Assuming only $\lambda \leq 1$, and since $\lambda(1 + \mu)^2/4 = \mu$, we have $\lambda^{(2)} \leq \mu^p(2/(1 + \mu^p))^2$, which implies $\mu^{(2)} \leq \mu^p$. QED

6.C. Back Substitution

Error propagation in the back substitution for p-fold reduction depends on the particular implementation. In the simplest form

$$x_{kp-j} = \varphi_{kp-j} - \alpha_{kp-j}\, x_{kp-p} - \gamma_{kp-j}\, x_{kp},$$

errors in x_{kp-j} depend on newly generated rounding errors and propagated errors from x_{kp-p} and x_{kp}. We can write the computed solution as $y_{kp-j} = x_{kp-j} + \xi_{kp-j}$ to find

$$\xi_{kp-j} = -\alpha_{kp-j}\,\xi_{kp-p} - \gamma_{kp-j}\,\xi_{kp} + \epsilon_{kp-j}.$$

It follows from the analysis of the Gauss-Jordan algorithm (Corollary 3.5) and from explicit formulae for α_{kp-j}, γ_{kp-j}, that $\|\alpha_{kp-j}\| \le \|B[A]\|^{p-j}$, $\|\gamma_{kp-j}\| \le \|B[A]\|^{j}$ if $\|B[A]\| < 1$, which leads to

$$\|\xi_{kp-j}\| \le \|B[A]\|^{p-j}\|\xi_{kp-p}\| + \|B[A]\|^{j}\|\xi_{kp}\| + \|\epsilon_{kp-j}\|.$$

Errors in a newly computed component therefore depend most strongly on errors in its closest back-substituted component, and the "influence" decreases as the distance between components increases. Moreover, we have

$$\|\xi_{kp-j}\| \le \|B[\tilde{A}]\| \max(\|\xi_{kp-p}\|, \|\xi_{kp}\|) + \|\epsilon_{kp-j}\|.$$

If ϵ is a bound on errors introduced at each stage, it follows that errors in the final computed result will be bounded by $\epsilon \log_p N$.

Slightly different results prevail in the back substitution

$$x_{kp-1} = \varphi_{kp-1} - \alpha_{kp-1}\,x_{kp-p} - \gamma_{kp-1}\,x_{kp}$$

for $j = 2,\ldots,p-1$

$$x_{kp-j} = F_{k,p-j} - A_{k,p-j}\,x_{kp-p} - C_{k,p-j}\,x_{kp-j+1}$$

Errors in the computed solution $y_{kp-j} = x_{kp-j} + \xi_{kp-j}$ obey the rule

$$\xi_{kp-1} = -\alpha_{kp-1}\,\xi_{kp-p} - \gamma_{kp-p}\,\xi_{kp} + \epsilon_{kp-1}$$

for $j = 2,\ldots,p-1$

$$\xi_{kp-j} = -A_{k,p-j}\,\xi_{kp-p} - C_{k,p-j}\,\xi_{kp-j+1} + \epsilon_{kp-j},$$

so the errors in x_{kp-j} depend on propagation from x_{kp-p} and x_{kp-j+1} rather than x_{kp-p} and x_{kp}. This implies a linear growth as in the LU factorization in conjunction with logarithmic growth as in odd-even reduction. If we again suppose that $\|\epsilon_{kp-j}\| \leq \epsilon$, with $\|\xi_{kp-p}\|, \|\xi_{kp}\| \leq \xi$, it follows that

$$\|\xi_{kp-1}\| \leq \|B[\tilde{A}]\|\,\xi + \epsilon,$$
$$\|\xi_{kp-j}\| \leq \|B[\tilde{A}]\| \max(\xi, \|\xi_{kp-j+1}\|) + \epsilon$$

which leads to $\|\xi_{kp-j}\| \leq \xi + (p-1)\epsilon$. The end result is that errors in the final computed result will be bounded by $\epsilon p \log_p N$.

7. SEMIDIRECT METHODS

Briefly, a semidirect method is a direct method that is stopped before completion. In common usage, a direct method for the solution of $Ax = v$ computes x exactly in a finite number of steps if exact arithmetic is used, while an iterative method computes a sequence of approximations $x^{(i)}$ converging to x. The semidirect methods* are intermediate on this scale, producing an approximation to x in a finite number of steps, while an additional finite number of steps could produce x exactly. During the computation, intermediate results are generated which approximate the solution. If the error is small enough then the algorithm is terminated prematurely and an approximate solution is returned.

The method of conjugate gradients is an example of a semidirect method since it can produce a good approximation long before its usual termination [R2]. The accelerated Parallel Gauss iteration ([H4] and Section 8.E) is another example, as is Malcolm and Palmer's convergent LU factorization [M2] of $A = (a,b,a)$, a and b positive definite with $\rho(b^{-1}a) < \frac{1}{2}$.

Here we consider semidirect methods based on the quadratic convergence properties of the odd-even methods. This will generalize various results obtained by Hockney [H5], Stone [S2] and Jordan [J1]. Buzbee [B17] discusses a similar technique based on the Buneman algorithm for Poisson's equation (cf. Theorem 4.3).

*The name semi-iterative has already been appropriated for other purposes [V3]. Stone [S2] initiated the name semidirect.

7.A. Incomplete Elimination

Let us first consider odd-even elimination for $Ax = v$, which is the process $H_1 = A$, $v^{(1)} = v$, $H_{i+1} = (I + C[H_i])H_i$, $v^{(i+1)} = (I + C[H_i])v^{(i)}$. In Theorem 4.1 we proved that if $\|B[A]\| < 1$ then $\|B[H_{i+1}]\| \le \|B[H_i]^2\|$, so that the off-diagonal blocks decrease quadratically in magnitude relative to the diagonal blocks. This motivates the approximate solution $z^{(i)} = D[H_i]^{-1}v^{(i)}$. Because we halt the process before completion, we call this technique incomplete odd-even elimination.

Theorem 7.1. $\|x - z^{(i)}\| / \|x\| \le \|B[H_i]\|$.

Proof. Since $v^{(i)} = H_i x = D[H_i](I - B[H_i])x$, we have $z^{(i)} = D[H_i]^{-1}v^{(i)} = x - B[H_i]x$. QED

Thus the B matrix determines the relative error in the simple approximation $z^{(i)}$. If $\beta = \|B[A]\| < 1$, $0 < \varepsilon < 1$, $m = \lceil \log_2 N \rceil$, then

$$(7.1)\quad k = 1 + \max\left(0, \min\left(m, \left\lceil \log_2\left(\frac{\log_2 \varepsilon}{\log_2 \beta}\right)\right\rceil\right)\right)$$

guarantees $\|B[H_k]\| \le \varepsilon$. For example, if $\beta = \frac{1}{2}$, $\varepsilon = 2^{-20} \doteq 10^{-6}$, then $k = 6$ and we should have $N > 32$ for incomplete odd-even elimination to be applied. In fact, $\|B[H_k]\| \le \beta^{2^5} = 2^{-32} \doteq 10^{-9.6}$, so the results of the incomplete elimination would be much better than required.

In the case $n = 1$, $a_j = c_j = a$, $b_j = b$, we have $\|B[H_{i+1}]\| = \|B[H_i]^2\| / (2 - \|B[H_i]^2\|)$. Since $x^2/2 < x^2/(2 - x^2) < x^2$ for $0 < x < 1$, k must be larger than

$$(7.2)\quad 1 + \log_2\left(\frac{\log_2(\varepsilon/2)}{\log_2(\beta/2)}\right)$$

in order to have $\| B[H_k] \| \le \varepsilon$. (By induction, $\beta^{(k)} > 2(\beta/2)^{2^{k-1}}$, and if $2(\beta/2)^{2^{k-1}} > \varepsilon$ then $\beta^{(k)} > \varepsilon$.) A more refined lower bound is obtained by defining $\tau_1 = \beta$, $\tau_{n+1} = \tau_n^2/(2 - \tau_n^2)$, and computing the largest k such that

(7.3) $\tau_k > \varepsilon$.

Since $B[H_i]$ is the matrix associated with the block Jacobi iteration for $H_i x = v^{(i)}$, when $\| B[H_k] \|$ is small we can use this iteration to improve the estimate $z^{(k)}$. In fact, the choice $z^{(k)} = D[H_k]^{-1}$ is the iterate following the initial choice $z^{(k)} = 0$. If $z^{(k,0)}$ is an approximation to x and $z^{(k,i+1)} = B[H_k]z^{(k,i)} + D[H_k]^{-1}v^{(k)}$ then $\| x - z^{(k,\ell)} \| \le \| B[H_k]^{\ell} \| \, \| x - z^{(k,0)} \|$. k and ℓ may now be chosen to minimize computation time subject to the constraint $\beta^{2^{k-1}\ell} \le \varepsilon$. Any of the other standard iterative methods may be used in place of the Jacobi iteration; for related results using the theory of non-negative matrices and regular splittings, see [H1].

We have remarked earlier that if the tridiagonal matrix $A = (a_j, b_j, c_j)$ is irreducibly diagonally dominant and $\beta^{(i)} = \| B[H_i] \|$ then $\beta^{(1)} \le 1$ and $\beta^{(i+1)} \le \beta^{(i)2}$. This is a case of practical importance in differential equations. However, the example $A = (-1, 2, -1)$, for which $\beta^{(i)} = 1$, $1 \le i \le m-1$, $\beta^{(m)} = 0$, shows that strict inequality is necessary for incomplete elimination to be effective.

Indeed, when $\beta^{(1)}$ is very close to 1, as will be the case for most second-order boundary value problems, the fact that we have strict inequality does not imply that $\beta^{(i)}$ will be small for $i < m$. With regard to the particular approximation $z^{(i)} = D[H_i]^{-1}v^{(i)}$, $z_j^{(i)}$ depends only on v_ℓ, $j - (2^{i-1} - 1) \le \ell \le j + (2^{i-1} + 1)$. Thus for $N = 2^m - 1$, $i < m$, the center point $z^{(i)}_{2^{m-1}}$ will be independent of the boundary data.* Although this is not the only possible approximation, complete odd-even elimination will be necessary to ensure convergence to the continuous solution.

7.B. Incomplete Reduction

The incomplete odd-even reduction algorithm is analagous to incomplete odd-even elimination. Complete reduction generates a sequence of block tridiagonal systems $A^{(i)}x^{(i)} = w^{(i)}$, $i = 1,\ldots,m$, $m = \lceil \log_2 N + 1 \rceil$, solves $A^{(m)}x^{(m)} = w^{(m)}$ exactly for $x^{(m)}$, and back substitutes to finally obtain $x = x^{(1)}$. Incomplete reduction stops with $A^{(k)}x^{(k)} = w^{(k)}$ for some k between 1 and m, solves this system approximately for $y^{(k)} \approx x^{(k)}$, and back substitutes $y^{(k)}$ to obtain $y = y^{(1)} \approx x^{(1)}$.

We remind the reader of observations made in Section 4.D, especially that roughly half the actual work is performed in the first reduction, one quarter in the second, and so forth. Thus the savings in incomplete reduction would be much less than those in incomplete elimination, where the work per step is roughly constant. For more details, see Section 10.C.

In contrast to incomplete elimination, the incomplete reduction must deal with error propagation in the back substitution. We have already shown that roundoff errors in complete reduction are well-behaved, and can only grow at most logarithmically. But since the approximation errors quite possibly might be larger than the roundoff errors, a different analysis is needed. We will see that a number of interesting properties result, including no growth of errors under certain assumptions about roundoff,

*This observation is due to an anonymous referee of [H2].

and a superconvergence-like effect due to damping of errors.

An initial approximation $y^{(k)}$ may be computed from $A^{(k)}$ and $w^{(k)}$ according to our analysis of incomplete elimination. The choice $y^{(k)} = D[A^{(k)}]^{-1}w^{(k)}$ is a simple one, and we have $\|x^{(k)}-y^{(k)}\|/\|x^{(k)}\| \le \|B[A^{(k)}]\|$. The parameter k should be determined both for computational economy and for a suitable approximation. Here the quadratic decreases of $B[A^{(i)}]$ are important and useful in some (but certainly not all) situations.

Regardless of the source of the approximation $y^{(k)}$, let us suppose that $y_j^{(k)} = x_j^{(k)} + \delta_j^{(k)}$. In the back substitution, $y^{(i)}$ is computed from $y^{(i+1)}$ by

$$y_{2j}^{(i)} = y_j^{(i+1)}$$

$$y_{2j-1}^{(i)} = (b_{2j-1}^{(i)})^{-1}(w_{2j-1}^{(i)} - a_{2j-1}^{(i)} y_{j-1}^{(i+1)} - c_{2j-1}^{(i)} y_j^{(i+1)}) + \epsilon_{2j-1}^{(i)},$$

with $\epsilon_{2j-1}^{(i)}$ representing rounding errors. We then have

$$\delta_{2j}^{(i)} = \delta_j^{(i+1)}$$

$$\delta_{2j-1}^{(i)} = \epsilon_{2j-1}^{(i)} - (b_{2j-1}^{(i)})^{-1}(a_{2j-1}^{(i)} \delta_{j-1}^{(i+1)} + c_{2j-1}^{(i)} \delta_j^{(i+1)}).$$

Theorem 7.2. If $\|\epsilon^{(i)}\| \le (1 - \|B[A^{(i)}]\|)\ \|\delta^{(i+1)}\|$ then $\|\delta^{(i)}\| = \|\delta^{(i+1)}\|$.

Proof. By construction,

$$\delta^{(i)} = \begin{pmatrix} \epsilon_1^{(i)} \\ \delta_1^{(i+1)} \\ \epsilon_3^{(i)} \\ \delta_2^{(i+1)} \\ \vdots \end{pmatrix} + B[A^{(i)}] \begin{pmatrix} 0 \\ \delta_1^{(i+1)} \\ 0 \\ \delta_2^{(i+1)} \\ \vdots \end{pmatrix}.$$

As in the analysis of roundoff error propagation for complete odd-even reduction, we have

$$\|\delta^{(i)}\| \le \max(\|\delta^{(i+1)}\|, \|\epsilon^{(i)}\| + \|B[A^{(i)}]\| \|\delta^{(i+1)}\|).$$

From this expression and the hypothesis we have $\|\delta^{(i)}\| \le \|\delta^{(i+1)}\|$. But, also by construction, $\|\delta^{(i)}\| \ge \|\delta^{(i+1)}\|$. QED

This theorem requires some explanation. First suppose that no rounding errors are committed, so that $\epsilon^{(i)} = 0$ and all the error in $y^{(1)}$ is derived from the approximation error in $y^{(k)}$. It now follows from $\|B[A]\| < 1$ that this error will not grow during the back substitution. From $\|x^{(k)}\| \le \|x\|$, we conclude that $\|x-y\| / \|x\| \le \|x^{(k)}-y^{(k)}\| / \|x^{(k)}\|$. In fact, it is easy to see that while $\|x - y\| = \|x^{(k)} - y^{(k)}\|$, many vector components of x-y will be much smaller in size than $\|x - y\|$. In Figure 7.1 we illustrate the errors by component for the tridiagonal system $(-1, 4, -1)x = v$, $N = 31$, $k = m-1 = 4$, v chosen so that $x_j = 1$, and $y_j^{(k)} = b_j^{(k)-1} w_j^{(k)}$. In the absence of rounding errors, the back substitution quickly damps out approximation errors, creating a very pleasing superconvergence-like effect.

Now suppose that rounding errors are committed and we have the upper bound $\|\epsilon^{(i)}\| \le \epsilon$. If $\|\delta^{(k)}\| \le \gamma\epsilon$ for some moderate constant γ then we have essentially solved the system $A^{(k)}x^{(k)} = w^{(k)}$ exactly, and the error propagation analysis is the same as for complete reduction. That is, we have $\|\delta^{(i)}\| \le (k - i + \gamma)\epsilon$, $1 \le i \le k$.

The remaining case is when $\|\delta^{(k)}\| >> \epsilon$. Here the semidirect method must be considered in competition with an iterative method. The hypothesis of Theorem 7.2, $\|\delta^{(k)}\| (1 - \|B[A^{(k)}]\|) \ge \epsilon$, should be viewed as

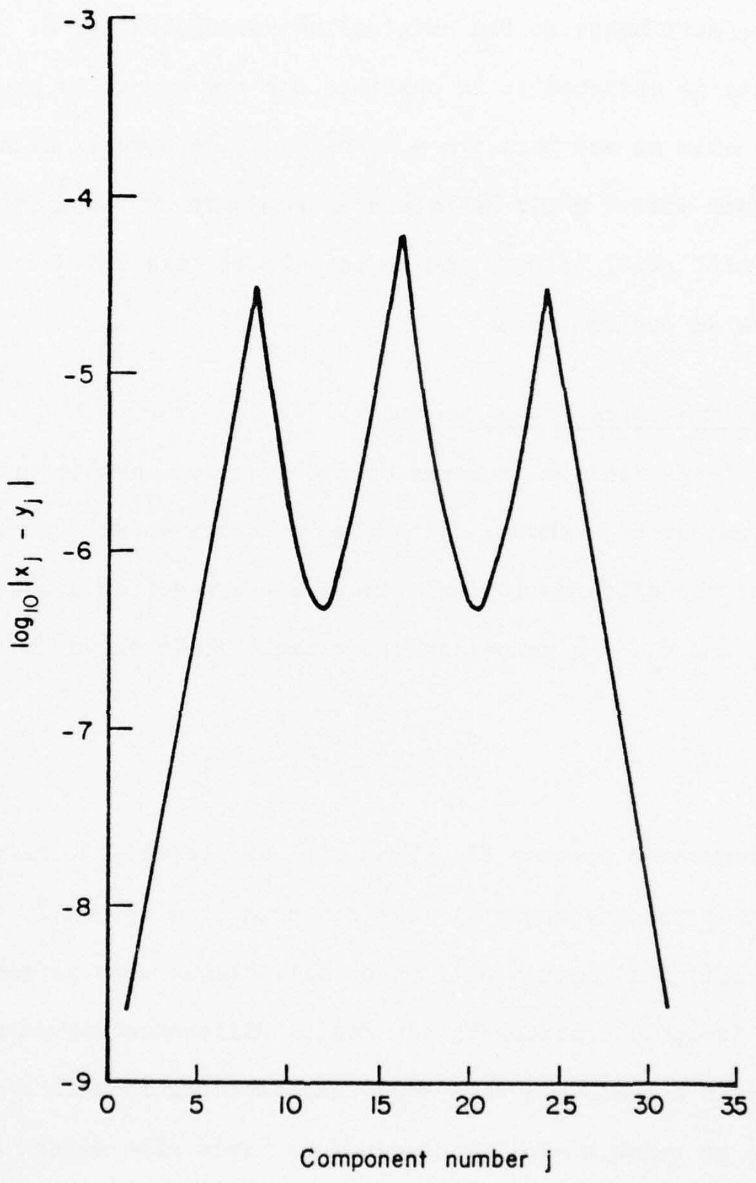


Figure 7.1. $\log_{10} |x_j - y_j|$ vs. j.

an attempt to formalize the condition $\|\delta^{(k)}\| \gg \varepsilon$. As long as the hypothesis remains true, the difference (in norm) between $x^{(i)}$ and $y^{(i)}$ can entirely be attributed to the original approximation error. As soon as the hypothesis is violated it is possible for the errors to grow, but we will still be able to say that $\|x - y\| = \|\delta^{(k)}\| + O(k\varepsilon)$. Although the superconvergence effect would be moderated somewhat by roundoff, its qualitative effects will still be felt and we can expect this bound on accumulated error to be an overestimate.

7.C. Applicability of the Methods

We close with some remarks on the practical application of the quadratic semidirect methods. In Buzbee's analysis [B17] of the Buneman algorithm for the differential equation $(-\Delta + d)u = f$ on a rectangle with sides σ_1 and σ_2, d a nonnegative constant, it is shown that if

$$\frac{\sigma_1}{\sigma_2}\,[d\sigma_2^2 + \pi^2]^{\frac{1}{2}} \gg 1$$

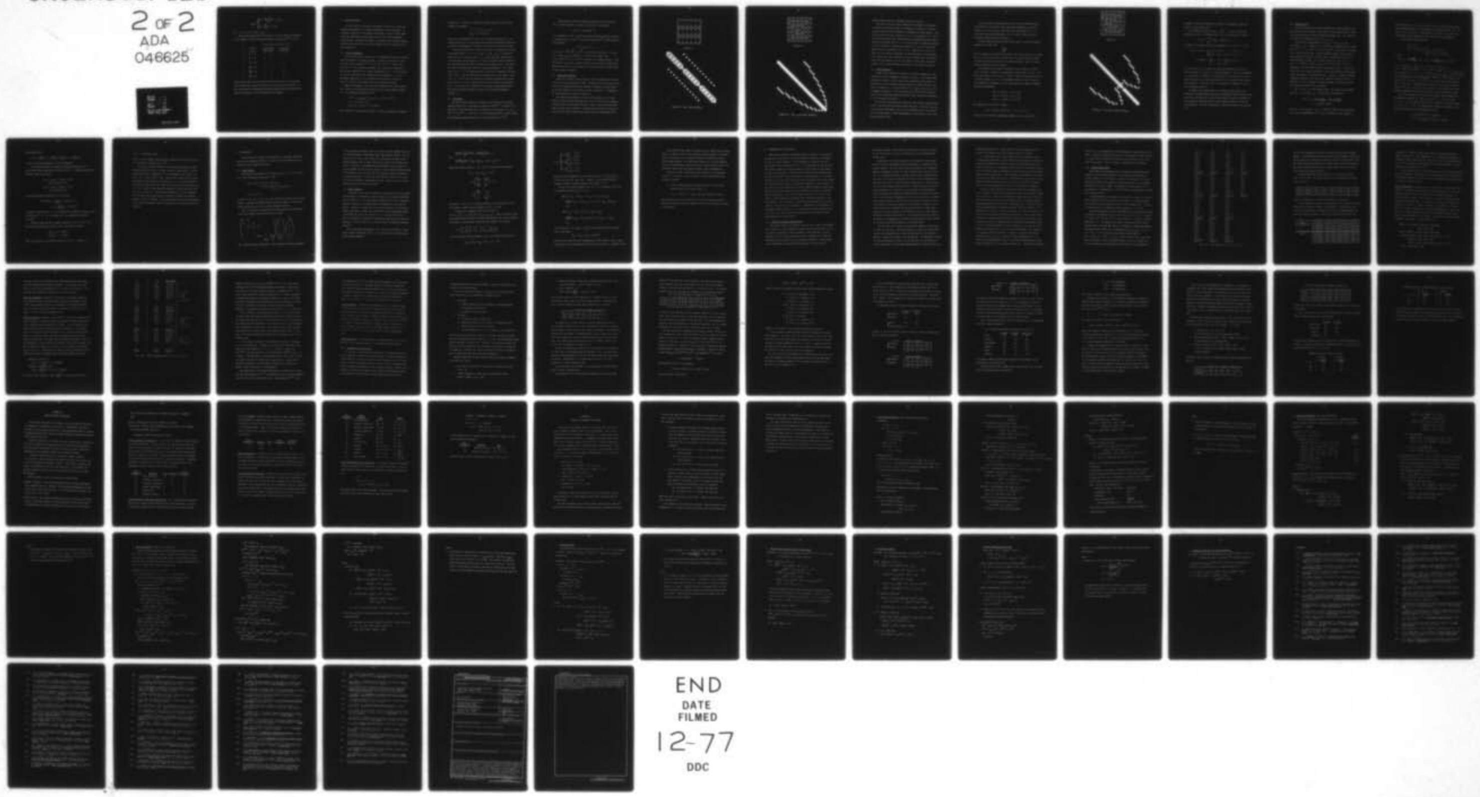
then a truncated Buneman algorithm will be useful. Geometrically this requires that the rectangle be long and thin if d is small, and hence the block tridiagonal matrix will have small blocks when properly ordered.

By directly considering the finite difference approximation to Poisson's equation on a rectangle with equal mesh spacing in both directions it is possible to reach a similar conclusion; this also serves as a simple model for more general elliptic equations. With M nodes in one direction and N in the other, $M < N$, the matrix A is $(-I_M, P_M, -I_M)_N$, $P_M = (-1, 4, -1)_M$. It is readily verified that $\|B[A]\| = 2\|Q_M\|$, $Q_M = P_M^{-1}$,

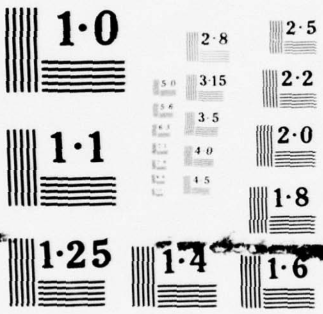
AD-A046 625 CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 12/1
DIRECT AND ITERATIVE METHODS FOR BLOCK TRIDIAGONAL LINEAR SYSTE--ETC(U)
APR 77 D E HELLER N00014-76-C-0370

UNCLASSIFIED NL

2 OF 2
ADA
046625



END
DATE
FILMED
12-77
DDC


1·0
2·8
2·5
3·15
2·2
5·0
5·6
3·5
2·0
1·1
4·0
4·5
1·8
1·25
1·4
1·6

$$\|R_n\| = \begin{cases} \frac{1}{2}\left(1 - \frac{D_m + D_{m-1}}{D_n}\right), & n = 2m, \\ \frac{1}{2}\left(1 - \frac{2D_m}{D_n}\right), & n = 2m+1, \end{cases}$$

$D_0 = 1$, $D_1 = 4$, $D_n = 4\, D_{n-1} - D_{n-2}$.

The following table gives $\| B[A] \|$ for $M = 1(1)6$, along with estimates on the number of odd-even reductions needed to obtain $\| B[A^{(i)}] \| < 10^{-8}$.

		k such that $\| B[A^{(k)}] \| < 10^{-8}$		
M	$\| B[A] \|$	lower bounds (7.2)	(7.3)	upper bound (7.1)
1	$\frac{1}{2} = .5$	5	5	6
2	$\frac{2}{3} \doteq .67$	6	6	7
3	$\frac{6}{7} \doteq .86$	6	7	8
4	$\frac{10}{11} \doteq .91$	6	7	9
5	$\frac{25}{26} \doteq .96$	6	8	10
6	$\frac{40}{41} \doteq .98$	6	8	11

These figures again lead to the conclusion that the best use of the semi-direct methods for elliptic equations will be with long thin regions. In the next section we consider some iterative methods which use matrix splittings and orderings to take advantage of this property.

8. ITERATIVE METHODS

In this section we consider the iterative solution of a finite difference approximation to a general nonseparable elliptic equation. Of particular interest will be methods which employ, as part of each iterative step, the direct or semi-direct solution of a block tridiagonal linear system. Continuing our emphasis on methods for a parallel computer, we want this system to be such that the method already considered can be used efficiently.

8.A. Use of Elimination

Let us first note that several of our theorems about block elimination have applications to iterative methods. Consider the system $Ax = v$ and the block Jacobi iteration $x^{(i+1)} = B[A]x^{(i)} + D[A]^{-1}v$. Suppose that $\|B[A]\| < 1$, so the iteration converges; this condition is satisfied for many elliptic equations with suitable partitioning [V3]. If the structure of A can be simplified by an elimination step, forming $A'x = v'$, then subsequent computations will be handled more efficiently, and since $\|B[A']\| \le \|B[A]\|$ the iteration will not converge any slower.

Indeed, convergence of the block Jacobi iteration will often be quite slow. When A is also positive definite the Jacobi semi-iterative method (J - SI or Chebychev acceleration) [V3]

$$x^{(i+1)} = \nu_{i+1}(B[A]x^{(i)} + D[A]^{-1}v - x^{(i-1)}) + x^{(i-1)},$$
$$\nu_1 = 1,\ \nu_2 = 2/(2 - \rho^2),$$
$$\nu_{i+1} = (1 - \rho^2\nu_i/4)^{-1},\ \rho = \rho(B[A]),$$

will converge at a much faster rate and is a natural candidate for parallel computation. If we use $\beta = \|B[A]\|$ as an upper bound on ρ and use the sequence of parameters

$$\nu_1^* = 1,\ \nu_2^* = 2/(2 - \beta^2),$$
$$\nu_{i+1}^* = (1 - \beta^2 \nu_i^*/4)^{-1},$$

then the J-SI iteration for A' will not be slower than that for A. By "preprocessing" $Ax = v$ we can expect that the actual computation time will be smaller; unfortunately a complete proof of this claim is not at hand.

Juncosa and Mullikin [J2] discuss the effect of elimination on iterative methods when $A = I - M$, $M \geq 0$, $m_{rr} = 0$, and $\sum_s m_{rs} \leq 1$ with strict inequality for at least one value of r. Their particular interest for elliptic boundary value problems is to order the boundary nodes followed by the interior nodes, and then to eliminate the boundary nodes. This would be useful for domains with curved or irregular boundaries. It is shown that, with $A' = I - M'$ representing the system after eliminating one point, $\rho(M') \leq \rho(M)$. Thus the point Jacobi iteration will not be slower. Also, if $G = (I - L[M])^{-1} U[M]$ is the matrix describing the Gauss-Seidel iteration, then $\rho(G') \leq \rho(G)$. Similar results follow from work on G-matrices [B4]. As in the proof of Corollary 3.2 we obtain corresponding results for block elimination, though not under the more general assumption $\|B[A]\| < 1$.

8.B. Splittings

The block Jacobi iteration is created by "splitting off" the block diagonal portion of A, and using the iteration $D[A]x^{(i+1)} = (D[A]-A)x^{(i)} + v$. Many other iterative methods are also defined by a splitting $A = M_1 - M_2$, $M_1 x^{(i+1)} = M_2 x^{(i)} + v$. When M_1 is a block tridiagonal matrix, various direct and semidirect methods can be used to solve the system for $x^{(i+1)}$.

A particularly effective procedure discussed by Concus and Golub [C4] is readily adapted to parallel computation. The equation

$$\mathcal{L}u \equiv -\nabla \cdot (a(x,y)\nabla u) = f$$

on a rectangle R, $a(x,y) > 0$ and sufficiently smooth, undergoes a change of variable $w(x,y) = a(x,y)^{\frac{1}{2}} u(x,y)$ and scaling by $a^{-\frac{1}{2}}$ to yield the equivalent equation

$$-\Delta w + p(x,y)w = q$$

on R, $p(x,y) = a^{-\frac{1}{2}} \Delta(a^{\frac{1}{2}})$, $q = a^{-\frac{1}{2}}f$. The shifted iteration $(-\Delta + k)w_{n+1} = (k - p)w_n + q$, k a constant, is solved in the discrete form $(-\Delta_h + KI)w^{(n+1)} = (KI - P)w^{(n)} + Q$, where P is a diagonal matrix. Since the system $(-\Delta_h + KI)b = c$ may be solved by the fast Poisson methods and Chebychev acceleration is applicable, the overall method is highly parallel and rapidly convergent.

8.C. Multi-line Orderings

There are a number of more classical iterative schemes which involve solution of block tridiagonal systems. We begin with the multi-line iterations for a rectangular grid; the ordering of unknowns and partitioning for 2-line iteration on a 6X6 grid is shown in Figure 8.1. A diagram of the resulting matrix is shown in Figure 8.2.

The underlying technique of the multi-line methods is to divide a square or long and broad rectangle into a set of long and thin rectangles. A discrete form of the elliptic equation is then solved over each subrectangle as part of the iteration. By partitioning the grid into groups of a few lines we obtain systems that can be solved well by general block

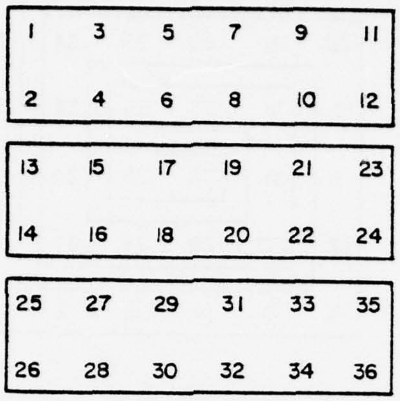


Figure 8.1

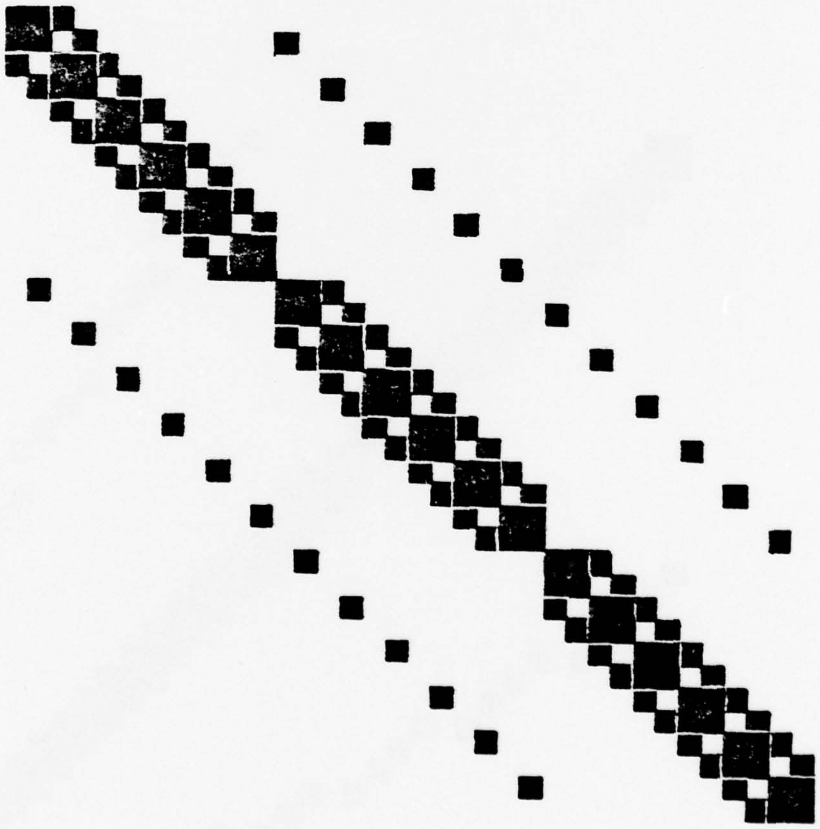


Figure 8.2 The 2-line ordering.

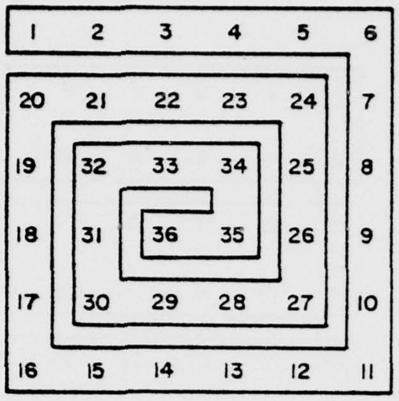


Figure 8.3

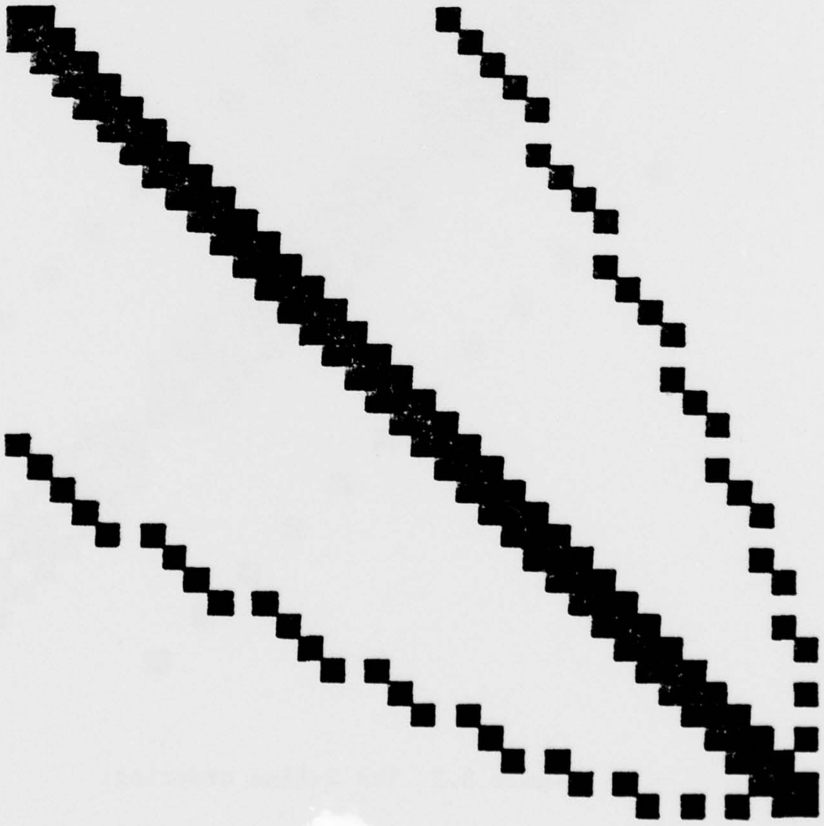


Figure 8.4. The single spiral ordering

methods, particularly the semidirect methods of Section 7.

The multi-line block Jacobi iteration with Chebychev acceleration is probably the simplest method to implement on a parallel or pipeline computer and should be considered as a benchmark for other methods. With an $n \times n$ grid and a k-line iteration, $n = kn'$, the splitting $A = M_1 - M_2$ is such that M_1 consists of n' decoupled block tridiagonal matrices, each of which has $k \times k$ blocks with $N = n$. By storing zeros in strategic locations M_1 may be regarded for computational purposes as one block tridiagonal matrix with $k \times k$ blocks and $N = nn' = n^2/k$. Lambiotte [L1] points out that only log n odd-even reductions are needed because of this construction, rather than log N reductions as for the general case. When incomplete odd-even reduction is used, the number of lines may be chosen according to the guidelines given in Section 7.C for Poisson's equation.

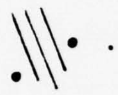
8.D. Spiral Orderings

Another useful ordering of unknowns for parallel computation is the single spiral order diagrammed in Figures 8.3 and 8.4. The spiral order essentially produces a very long and thin rectangular region, which is represented by the tridiagonal matrix forming the central portion of the matrix in Figure 8.4. In the splitting $A = M_1 - M_2$ we take M_1 to be the tridiagonal center of A. For an $n \times n$ grid M_1 is $n^2 \times n^2$ and has $\| B[M_1] \|$ about $\frac{1}{2}$; thus incomplete odd-even reduction can be used quite efficiently to solve systems involving M_1.

We note that the single spiral order has been used with another splitting to yield the block peripheral method [B9], [E2]. Here the blocks are of the periodic tridiagonal form, and the convergence rate of the peripheral block SOR method is shown experimentally to be similar to the 2-line block SOR method [B10].

The double spiral ordering (Figure 8.5, 8.6) is an attempt to combine features of the single spiral order with advantages of a 2-cyclic ordering. For an n×n grid, n odd, one spiral contains $\frac{1}{2}(n^2 + 2n - 1)$ points and the other contains $\frac{1}{2}(n^2 - 2n + 1)$ points. Multi-spiral or -peripheral orders can also be defined.

For the single and double spiral orders, when n is even the trailing submatrix of A looks like



The •'d elements of A should be included in the implicit part of the iteration, M_1. This can be done at little extra cost, and ensures that each row and column of M_2 has at most two non-zero elements. When n is odd a similar modification should be made.

The mapping from the natural ordering to the peripheral or single spiral ordering is defined by a permutation of the vector $T = (1,2,\ldots,n^2)$ into a new vector $S = (S(1),S(2),\ldots,S(n^2))$. With an n×n grid, point (i,j) is unknown number $(i-1)n + j$ in the natural ordering. Given T, two vectors forming the coordinates (i,j) of each unknown are computed by taking quotient and remainder:

$$q(k) = \lfloor (k-1)/n \rfloor, \quad (1 \le k \le n^2)$$
$$i(k) = 1 + q(k), \quad (1 \le k \le n^2)$$
$$j(k) = k - nq(k), \quad (1 \le k \le n^2).$$

For simplicity let $n = 2m-1$. Define

$$p(i,j) = \min(i,j,\ 2m-i,\ 2m-j), \quad (1 \le i,\ j \le n).$$

The point (i,j) is part of peripheral number $p(i,j)$, $1 \le p(i,j) \le m$.

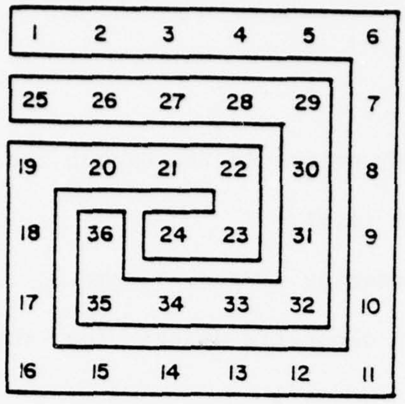


Figure 8.5

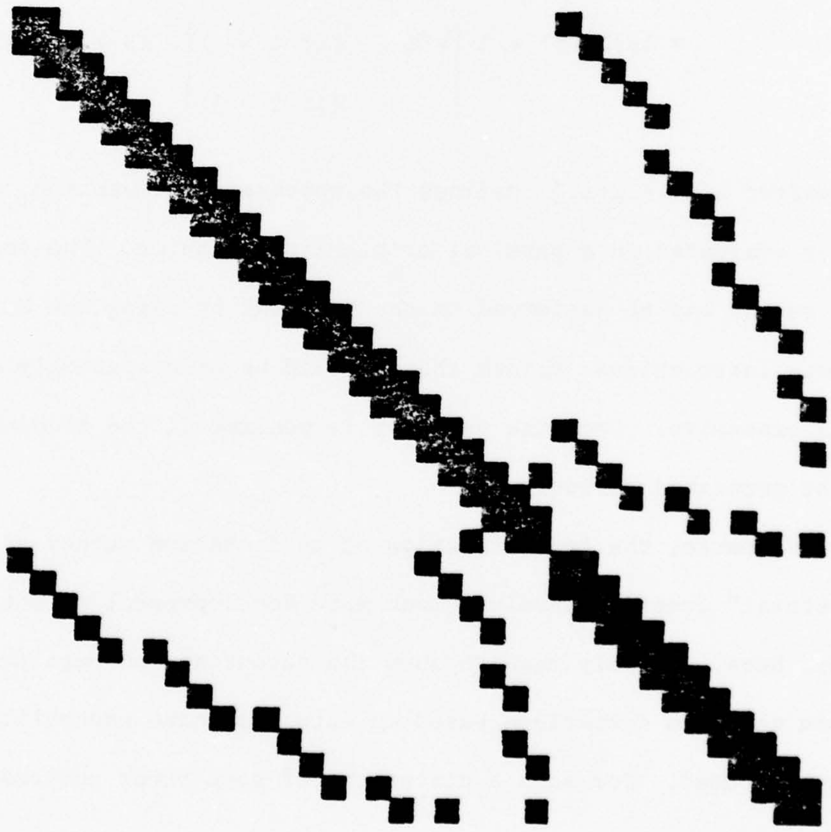


Figure 8.6. The double spiral ordering

Peripheral p has 8(m-p) points in it, except for peripheral m which contains the single point (m,m).

The peripheral ordering is per. 1, per.2, ..., per. m, where each peripheral is taken in clockwise order: top, rhs, bottom, lhs. Define

$$f(i,j) = \sum_{k=1}^{p-1} 8(m-k) = 4(2m-p)(p-1),$$

the number of elements contained in peripherals 1,2,...,p(i,j) - 1, and

$$g(i,j) = f(i,j) + \begin{cases} (i-p) + (j-p) + 1 & \text{if } i \le j, \\ 8(m-p) - (i-p) - (j-p) + 1 & \text{if } i > j \end{cases}$$

$$= 4p(2m-p) + 1 \left\{ \begin{array}{ll} -8m + & (\text{if } i \le j) \\ \quad - & (\text{if } i > j) \end{array} \right\} (2p + i + j).$$

The vector S(k) = g(i,j) defines the necessary permutation, and can be easily evaluated on a parallel or pipeline computer. The actual reordering of a vector may be performed on the CDC STAR by using the built-in vector permute instructions, though these should be used sparingly since they are quite expensive. Once the vector S is generated, the elements of A and v may be generated directly.

Of course, the implementation of an iterative method based on an "unnatural" ordering involves much more developmental attention than is presented here. We only mean to show the potential for parallel computation. A more complete comparison based on actual machine capabilities has not been performed. For such a discussion of some other methods, see Lambiotte [L1].

8.E. Parallel Gauss

We close our section on iterative methods for block tridiagonal systems with some brief remarks on the Parallel Gauss iteration due to Traub [T2] with extensions by Heller, Stevenson and Traub [H4]. For simplicity we consider the normalized form $A = (a_j, I, c_j)$.

In the block LU factorization $A = (\ell_j, I, 0)(0, d_j, c_j)$, let $D = \text{diag}(d_1,\dots,d_N)$. Since $D = I - L[A]D^{-1}U[A]$, a natural parallel iteration is $D^{(i)} = I - L[A](D^{(i-1)})^{-1}U[A]$, where $D^{(0)}$ is some initial estimate of D. This forms the first of three iterations making up the Parallel Gauss method; the other two parts are Jacobi iterations for the L and U block bidiagonal systems. It is shown in [H4] that, with $\lambda(A) \le 1$, $\mu(A) = (1 - \sqrt{1-\lambda})/(1 + \sqrt{1-\lambda})$, we have $\|D - D^{(i)}\| \le \mu \|D - D^{(i-1)}\|$. Techniques for reducing the constant μ are given in [H4]. As Stone [S2] points out, this linear convergence is much less attractive than the quadratic convergence of odd-even reduction. However, in certain situations (parabolic equations or the multi-line iterations discussed here) it is necessary to solve a sequence of related problems, and this would hopefully provide suitable initial estimates.

Let $A' = (a'_j, I, c'_j)$ and suppose that D' has already been computed. Since $d'_j - d_j = a_j d^{-1}_{j-1} c_{j-1} - a'_j(d'_{j-1})^{-1}c'_{j-1}$, by Theorem 3.4

$$\|d'_j - d_j\| \le (\tfrac{\lambda}{4})(\frac{2}{1 + \sqrt{1-\lambda}}) + (\tfrac{\lambda'}{4})(\frac{2}{1 + \sqrt{1-\lambda'}})$$
$$= 1 - \tfrac{1}{2}(\sqrt{1-\lambda} + \sqrt{1-\lambda'}).$$

Thus $\|D' - D\| = \max_j \|d'_j - d_j\| \le 1 - \frac{1}{2}(\sqrt{1-\lambda} + \sqrt{1-\lambda'})$, and we may take D' as an initial approximation to D. Now, the Parallel Gauss iteration is most useful when λ and λ' are not too close to 1, and in this case good initial approximations will be available by the above bound on $\|D' - D\|$. When λ and λ' are close to 1 the bound will be large, but in this case the iteration should probably not be used due to slow convergence.

For tridiagonal matrices we have

$$\begin{aligned} d'_1 - d_1 &= 0, \\ d'_j - d_j &= a_j(d'_{j-1})^{-1}(d'_{j-1} - d_{j-1})d^{-1}_{j-1}\, c_{j-1} \\ &\quad + (d'_{j-1})^{-1}(a_j\, c_{j-1} - a'_j\, c'_{j-1}), \end{aligned}$$

so that

$$|d'_j - d_j| \le \frac{1 - \sqrt{1-\lambda}}{1 + \sqrt{1-\lambda'}}\, |d'_{j-1} - d_{j-1}| + \frac{2}{1 + \sqrt{1-\lambda'}}\, |a_j c_{j-1} - a'_j c'_{j-1}|.$$

Let $\Delta = \max_j |a_j c_{j-1} - a'_j c'_{j-1}| \le (\lambda + \lambda')/4$, $\delta_j = |d'_j - d_j|$, $a = (1 - \sqrt{1-\lambda})/(1 + \sqrt{1-\lambda'})$, $b = 2/(1 + \sqrt{1-\lambda'})$. Then $\delta_1 = 0$, $\delta_j \le a\, \delta_{j-1} + b$, so by induction $\delta_j \le (1 + a + \ldots + a^{j-2})b < b/(1-a)$, or $\delta_j \le 2\Delta/(\sqrt{1-\lambda} + \sqrt{1-\lambda'})$. By the intermediate value theorem $(\sqrt{1-\lambda} + \sqrt{1-\lambda'})/2 = \sqrt{1-\lambda^*}$ for some λ^* between λ and λ', so that $\delta_j \le \Delta/\sqrt{1-\lambda^*}$. Again we have the situation where, if λ^* is close to 1, a large upper bound results, but here the iteration should not be used. When λ^* is not so close to 1 the bound may give more complete information than the one derived for block tridiagonal systems.

Let us now consider the utility of the Parallel Gauss iteration for the parabolic equation $u_t = (au_x)_x$, $a = a(t,x) > 0$, $u = u(t,x)$, with $0 \le x \le 1$, $0 \le t$, and $u(0,x)$, $u(t,0)$, $u(t,1)$ specified. Let $u^i_j \approx u(ik,jh)$, $k = \Delta t$, $h = \Delta x$, $\rho = k/h^2$. The Crank-Nicolson method for advancing to the next time step must solve the tridiagonal system

$$\begin{aligned} (I + \tfrac{\rho}{2} P^{i+1})u^{i+1} &= (I - \tfrac{\rho}{2} P^i)u^i, \\ P^i &= (-\, a^i_{j-1/2},\, a^i_{j-1/2} + a^i_{j+1/2},\, -\, a^i_{j+1/2}). \end{aligned}$$

We therefore have

$$A = (-\rho a^{i+1}_{j-1/2},\ 1 + \rho(a^{i+1}_{j-1/2} + a^{i+1}_{j+1/2})/2,\ -\rho a^{i+1}_{j+1/2}/2);$$

let us now drop the superscript i+1 for convenience.

A will always be strictly diagonally dominant because $a(t,x) > 0$, and we would like to know when we also have $\lambda(A) \le 1$. Using the mean value theorem, define $\bar{x}_j$ and $\tilde{x}_j$ by

$$\begin{aligned} a_{j-1/2} + a_{j+1/2} &= 2a_{j-1/2} + h\,\bar{b}_j, \\ \bar{b}_j &= a_x((i+1)k,\ \bar{x}_j), \\ a_{j-3/2} + a_{j-1/2} &= 2a_{j-1/2} - h\,\tilde{b}_j, \\ \tilde{b}_j &= a_x((i+1)k,\ \tilde{x}_j). \end{aligned}$$

After some algebraic manipulation it is seen that

$$\begin{aligned} \lambda(A) = \max_j &(1 + \frac{1}{\rho a_{j-1/2}}(1 + \frac{\rho}{2} h\,\bar{b}_j))^{-1} \\ &\times (1 + \frac{1}{\rho a_{j-1/2}}(1 - \frac{\rho}{2} h\,\tilde{b}_j))^{-1}. \end{aligned}$$

In order to guarantee $\lambda(A) \le 1$ it is sufficient to assume that h and k satisfy $\frac{\rho}{2} h|a_x(t,x)| < 1$. If $m_1 = \max|a_x(t,x)|$ this becomes the condition $m_1 k \le 2h$.

Suppose we want the more stringent condition $\lambda(A) \le (1 + c)^{-2}$, $c \ge 0$. This may be satisfied if we choose h and k such that, for each j,

$$\begin{aligned} c\rho a_{j-1/2} &\le 1 + \rho h\bar{b}_j/2, \\ c\rho a_{j-1/2} &\le 1 - \rho h\tilde{b}_j/2. \end{aligned}$$

With $m_0 = \max a(t,x)$, a sufficient condition is $c\rho m_0 \le 1 - \rho h m_1/2$, or

(8.1) $k \le 2h^2/(2cm_0 + hm_1)$.

For $c = 0$, (8.1) reduces to the previous condition on h and k, while for $c = h$ we have $k \le 2h/(2m_0 + m_1) = O(h)$.

Now, one of the best features of the Crank-Nicolson method is that k may be chosen to be $O(h)$. We have seen that a small value of c still allows this choice. Conversely, if $k = rh$ satisfies (8.1) and $rm_1 < 2$, then $c \le h(1 - rm_1/2)/(rm_0) = O(h)$. When we try to pick h and k so that the Parallel Gauss method will be effective ($c \gg h$) we are necessarily restricted to small time steps. When h and k are chosen in the usual way ($k = O(h)$), Parallel Gauss cannot be expected to be very effective since $\lambda(A)$ will be close to 1. Thus the method appears to be less attractive for this particular application than originally believed, and odd-even reduction would be preferred. Once several reductions have been performed, λ and μ may be sufficiently small to warrant use of Parallel Gauss and its variations.

9. APPLICATIONS

In this section we discuss two applications of practical importance in which the assumption $\|B[A]\| < 1$ is not met; in the second case it is shown how to get around this problem.

9.A. Curve Fitting

Cubic spline interpolation of the data (x_i, f_i), $1 \le i \le N+1$, gives rise to the tridiagonal system [C6, p. 238]

$$\begin{aligned} h_i s_{i-1} &+ 2(h_i + h_{i-1})s_i + h_{i-1}s_{i+1} \\ &= 3(f[x_{i-1},x_i]h_i + f[x_i,x_{i+1}]h_{i-1}), \\ &\qquad 2 \le i \le N, \end{aligned}$$

$$h_i = x_{i+1} - x_i.$$

We have $A = (h_i,\ 2(h_i + h_{i-1}),\ h_{i-1})$ and $\|B[A]\| = \frac{1}{2}$, which is quite satisfactory. However, more general problems need not behave so nicely even though they might give rise to block tridiagonal matrices.

Cox [C7] presents an algorithm for least-squares curve fitting with piecewise polynomials which must solve the system

$$\begin{pmatrix} H_1 & C_1^T & & & & & \\ C_1 & 0 & C_2 & & & & \\ & C_2^T & H_2 & C_3^T & & & \\ & & \ddots & & \ddots & \ddots & \\ & & & C_{2m-3} & 0 & C_{2m-2} & \\ & & & & C_{2m-2}^T & H_m & \end{pmatrix} \begin{pmatrix} v_1 \\ \lambda_1 \\ v_2 \\ \vdots \\ \lambda_{m-1} \\ v_m \end{pmatrix} = \begin{pmatrix} b_1 \\ 0 \\ b_2 \\ \vdots \\ 0 \\ b_m \end{pmatrix}$$

The v vectors contain coefficients of the Chebychev polynomial expansion of the piecewise polynomial, and the λ vectors contain Lagrange multipliers derived through a constrained linear least-squares problem. The C_j's always have full rank, and if there are sufficiently many data points between adjacent pairs of knots then the H_j's will all be positive definite. In this case a straightforward band elimination may be used well when the number of knots and data points is large. When insufficient data is available the H_j's will not have full rank and pivoting is needed in the band elimination. It seems that the condition $\|B[A]\| < 1$ will not hold for any reasonable partitioning or data. Indeed, $B[A]$ is not even defined for the most obvious partition.

9.B. Finite Elements

In Section 2.A we noted that matrices constructed from finite element approximations to differential equations often do not satisfy assumptions such as $\|B[A]\| < 1$. Thus some other technique must be used to establish numerical stability in a direct method or convergence in an iterative method. Usually it is shown that A is positive definite. The following example, admittedly a simple one, shows how an easily implemented change of variables can yield a new system $A'x' = v'$ such that $\|B[A']\| < 1$. Even if the change of variables is not actually made, this will show that $\|B[A]\|_\alpha < 1$ for some norm $\|\cdot\|_\alpha$, and also serves to establish numerical stability.

Let us begin with the equation $-u'' = 1$ on the unit interval. Using Hermite cubics, Strang and Fix [S4, p. 59] derive the 4th order accurate finite element equations

$$(9.1)\quad \begin{aligned} &-\frac{6}{5}\left(\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2}\right) + \frac{1}{5}\left(\frac{u'_{j+1} - u'_{j-1}}{2h}\right) = 1, \\ &-\frac{1}{5}\left(\frac{u_{j+1} - u_{j-1}}{2h}\right) + \frac{1}{5}u'_j - \frac{1}{30}(u'_{j+1} - 2u'_j + u'_{j-1}) = 0 . \end{aligned}$$

Taking the natural choice $x_j = (u_j, u'_j)^T$, we obtain the block equation

$$ax_{j-1} + bx_j + cx_{j+1} = (1,0)^T$$

$$a = \begin{pmatrix} \frac{-6}{5h^2} & \frac{-1}{10h} \\ \frac{1}{10h} & \frac{-1}{30} \end{pmatrix} = c^T,$$

$$b = \begin{pmatrix} \frac{12}{5h^2} & 0 \\ 0 & \frac{4}{15} \end{pmatrix}.$$

The matrix $A = (a, b, c)$ is positive definite, but $\|B[A]\| = \frac{1}{4} + 3/(4h)$. The problem is simply that the unknowns are out of scale.

A second set of unknowns, which are in scale, is $x_j^T = (u_j - \frac{h}{2}u'_j, u_j + \frac{h}{2}u'_j)^T \approx (u_{j-1/2}, u_{j+1/2})^T$. Since the Hermite cubics can be derived from B-splines by coalescing adjacent nodes, these unknowns are actually a more natural choice than the original. We now have

$$x_j = \begin{pmatrix} u_j - \frac{h}{2}u'_j \\ u_j + \frac{h}{2}u'_j \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & h/2 \end{pmatrix} \begin{pmatrix} u_j \\ u'_j \end{pmatrix},$$

and, multiplying the first equation of (9.1) by h^2 and the second by h,

$$ax_{j-1} + bx_j + cx_{j+1} = (h^2 \cdot 1, h \cdot 0)^T,$$

$$(9.2)\quad \begin{cases} a = \frac{1}{2}\begin{pmatrix} -1 & -7/5 \\ 1/6 & 1/30 \end{pmatrix}, \\ b = \frac{1}{2}\begin{pmatrix} 12/5 & 12/5 \\ -8/15 & 8/5 \end{pmatrix}, \\ c = \frac{1}{2}\begin{pmatrix} -7/5 & -1 \\ -1/30 & -1/6 \end{pmatrix}. \end{cases}$$

The blocks are independent of h and we have $\|B[A]\| = 1$, with $\rho_k(B[A]) < 1$ for the first and last block rows. (Actually, this assumes Dirichlet boundary conditions; Neumann conditions give $\rho_k(B[A]) = 1$.)

For the more general equation $-u'' + Qu = f$, Q a constant, the following equations are derived [S4, p. 59]:

$$\frac{1}{30h}(-36\,u_{j-1} - 3h\,u'_{j-1} + 72\,u_j - 36\,u_{j+1} + 3h\,u'_{j+1})$$
$$+ \frac{Qh}{420}(54\,u_{j-1} + 13\,h\,u'_{j-1} + 312\,u_j + 54\,u_{j+1} - 13h\,u'_{j+1})$$
$$= F_j,$$
$$\frac{1}{30}(3\,u_{j-1} - h\,u'_{j-1} + 8h\,u'_j - 3\,u_{j+1} - h\,u'_{j+1})$$
$$+ \frac{Qh^2}{420}(-13\,u_{j-1} - 3h\,u'_{j-1} + 8h\,u'_j + 13\,u_{j+1} - 3\,h\,u'_{j+1})$$
$$= F'_j.$$

Again taking $x_j^T = (u_j - \frac{h}{2}u'_j,\ u_j + \frac{h}{2}u'_j)^T$ and suitably scaling the equations, they become

$$ax_{j-1} + bx_j + cx_{j+1} = (hF_j, F'_j)^T$$

where now the blocks are $O(h^2)$ perturbations of the blocks in (9.2). Omitting details, for h sufficiently small this gives $\|B[A]\| = 1 - \frac{5}{12}Qh^2 + O(h^4)$.

Since rescaling and change of variables in the linear system is equivalent to a change of basis in the original function subspace, the process should be applicable to more general differential equations solved by finite element methods. The underlying technique is to prove something for spline approximations, and show that the property carries over in the collapse of nodal points yielding the Hermite cubic approximations.

Strictly in terms of modifying linear systems by a block diagonal scaling, it is natural to ask if either of the following problems are solvable in general:

1. Given an arbitrary block tridiagonal matrix A, find a block diagonal matrix E such that $\|B[AE]\| < 1$.

2. In 1., if $\|B[A]\| < 1$, find E such that $\|B[AE]\| < \|B[A]\|$.

These problems are closely related to the optimal diagonal scaling of matrices to minimize condition numbers, and as such are outside the scope of this thesis.

10. IMPLEMENTATION OF ALGORITHMS

This section contains some general remarks on the choice of algorithm for particular applications, and a more detailed comparison of algorithms for use on a vector computer. In the comparison we assume uniform $n \times n$ block sizes, later specialized to $n = 2$. All algorithms will use the same data layout, assumed to be core-contained, and the execution time comparison will be in two parts: first for a very simplified and idealized model in which the machine vectors are defined as storage locations in arithmetic progression, and second for a model in which the machine vectors are defined as contiguous storage locations (arithmetic progression with increment = 1). The first model uses information for the CDC STAR-100 ignoring, among other things, the fact that machine vectors on STAR must satisfy the conditions for the second model. This double comparison allows us to pinpoint the effect of data manipulation required by the stringent requirement on vector operands. For a description of STAR facilities necessary for this discussion, see Section 2.B and Appendix A. Implementation of algorithms for tridiagonal linear systems, $n = 1$, is adequately discussed elsewhere ([H4], [J1], [L2], [M1], [S2]) and these references may be considered as background material for this section.

10.A. Choice of Algorithm (Generalities)

The salient point of our analysis has been that a number of algorithms may be applied to block tridiagonal linear systems that are likely to arise in practice. Sometimes it may be possible to take advantage of special features in order to reduce computing costs or estimate error. When the computing costs are high (such as one enormous problem or a large problem solved repeatedly) an extra effort must be made to choose the most efficient algorithm. This choice will be difficult to make in general, but we feel that it is not at all difficult to choose good algorithms in many cases.

Regardless of the computer involved, algorithms for three special problems should be implemented before proceeding to algorithms for more general problems. The first should deal with general block tridiagonal matrices restricted to blocks of a small uniform size, say no more than 8x8. On a standard sequential computer, band or profile methods are appropriate; on a k-parallel computer some form of block LU factorization should be used, and on a vector computer a combination of odd-even reduction and LU factorization should be used. Storage allocation and access, vector operations on a parallel or vector computer and implementation issues in general can be greatly simplified under the block size restriction. Options include implementation as a set of subroutines for specific block sizes, insertion of tests for use as a semidirect method, and special scalings or pivoting strategies for efficiency or stability. This restricted class of algorithms may be applied directly to small-block problems or used as a module in the iterative solution of large sparse problems as indicated in Section 8. The experience gained may or may not be useful for implementation of methods for moderate-sized blocks, which, in our opinion, should be treated in a different manner.

The best choice of a parallel algorithm for moderate-sized blocks is not clear at this time. These cases represent a cross-over between efficient vector implementations of band/profile methods and the odd-even methods. Either one is likely to be satisfactory. In Lambiotte's opinion [L1] the band methods are better for the CDC STAR, but this choice is heavily influenced by vector addressing restrictions on the STAR (cf. our remarks in Section 10.B). Other parallel or vector computers may not present such problems, and a more complete analysis is needed.

The third basic class of algorithms are the fast Poisson solvers, including generalizations. Recent studies have shown that many finite difference approximations to elliptic equations can be computed very efficiently by iterative or modification methods based on the Poisson solvers. There are several kinds of algorithms to choose from, and all have considerable inherent parallelism. Other algorithms are available for elliptic equations, but flexibility and ease of implementation suggest that Poisson-based algorithms should be among the first to be considered. Brandt's multi-level adaptive methods [B13] must also be seriously considered.

Finally, we note that matrix partitioning and block elimination is a useful technique for dealing with matrices so large that secondary storage must be used [R4]. Known as the hypermatrix scheme in other contexts, it has been extensively exploited by the ASKA system, a structural analysis package developed at the University of Stuttgart [F3]. Noor and Voigt [N1] discuss implementation of hypermatrix schemes on the CDC STAR-100. The method works as follows: suppose a partition π is given (see Section 2.A), and let A_π be the matrix A partitioned accordingly. Block sizes in ASKA are chosen to optimize transfers between storage levels, while the choice on STAR must also try to maximize vector lengths and hence minimize start-up costs. The usual situation is that a few blocks of A_π will be in main memory, while the bulk of A_π will be in secondary memory. For example, if π' is another partition such that π refines π', we would perform block elimination on $A_{\pi'}x_{\pi'} = v_{\pi'}$. When A_π is block tridiagonal so is $A_{\pi'}$. Since the data requirements for the LU or Cholesky factorization are then highly localized, this is a natural procedure [E3]. Moreover, the diagonal blocks of $A_{\pi'}$ will themselves be block tridiagonal, though possibly of low order. As noted in Sections 2.A and 3.A.2, if $\|B[A_{\pi}]\| < 1$ then $\|B[A_{\pi'}]\| \le \|B[A_{\pi}]\|$, and pivoting will be restricted to those blocks of $A_{\pi'}$ resident in main memory. This, of course, is precisely what is desired.

10.B. Storage Requirements

Data layout is an essential part of planning an algorithm's implementation on any parallel computer, since computational speed can be severely decreased if data cannot be accessed conveniently. On a vector computer this means that conceptual vectors must conform to the machine's definition of a vector operand. We consider some consequences of this requirement for implementation on the CDC STAR-100, assuming that the problem is core-contained.

Storage available on the STAR, for 64-bit words, consists of 256 registers and either 512K or 1M words of main memory. A sizable secondary disc storage is also available but will not be considered here. A vector on STAR consists of s consecutive storage locations, $1 \le s \le 64K$.

Our data arrangement for block tridiagonal linear systems with uniform nxn blocks is illustrated in fig. 10.1 for the case $n = 2$, $N = 7$. Define $m = \lceil \log_2 N \rceil$, $N^* = 2^m$, and if $N < N^*$ then extend $Ax = v$ with $N^* - N$ trivial equations $x_i = 0$. The matrix A and vector v can then be held in three storage vectors of length n^2N^* and one of length nN^*. Taking the subdiagonal blocks as an example, the n^2 components of a_k are $a_{ij,k}$, $1 \le i \le n$, $1 \le j \le n$, $1 \le k \le N^*$, and the storage vector $\underline{a}$ is defined by $\underline{a}((i - 1)N^*n + (j - 1)N^* + k) = a_{ij,k}$. Thus a total of $(3n^2 + n)N^*$ locations are needed to store the original system. We do not consider the cost of setting up the storage vectors since each of our algorithms will

a	b	c	v
$a_{11,1} = 0$	$b_{11,1}$	$c_{11,1}$	$v_{1,1}$
$a_{11,2}$	$b_{11,2}$	$c_{11,2}$	$v_{1,2}$
$a_{11,3}$	$b_{11,3}$	$c_{11,3}$	$v_{1,3}$
$a_{11,4}$	$b_{11,4}$	$c_{11,4}$	$v_{1,4}$
$a_{11,5}$	$b_{11,5}$	$c_{11,5}$	$v_{1,5}$
$a_{11,6}$	$b_{11,6}$	$c_{11,6}$	$v_{1,6}$
$a_{11,7}$	$b_{11,7}$	$c_{11,7} = 0$	$v_{1,7}$
$a_{11,8} = 0$	$b_{11,8} = 1$	$c_{11,8} = 0$	$v_{1,8} = 0$
$a_{12,1} = 0$	$b_{12,1}$	$c_{12,1}$	$v_{2,1}$
$a_{12,2}$	$b_{12,2}$	$c_{12,2}$	$v_{2,2}$
$a_{12,3}$	$b_{12,3}$	$c_{12,3}$	$v_{2,3}$
$a_{12,4}$	$b_{12,4}$	$c_{12,4}$	$v_{2,4}$
$a_{12,5}$	$b_{12,5}$	$c_{12,5}$	$v_{2,5}$
$a_{12,6}$	$b_{12,6}$	$c_{12,6}$	$v_{2,6}$
$a_{12,7}$	$b_{12,7}$	$c_{12,7} = 0$	$v_{2,7}$
$a_{12,8} = 0$	$b_{12,8} = 0$	$c_{12,8} = 0$	$v_{2,8} = 0$
$a_{21,1} = 0$	$b_{21,1}$	$c_{21,1}$	
$a_{21,2}$	$b_{21,2}$	$c_{21,2}$	
$a_{21,3}$	$b_{21,3}$	$c_{21,3}$	
$a_{21,4}$	$b_{21,4}$	$c_{21,4}$	
$a_{21,5}$	$b_{21,5}$	$c_{21,5}$	
$a_{21,6}$	$b_{21,6}$	$c_{21,6}$	
$a_{21,7}$	$b_{21,7}$	$c_{21,7} = 0$	
$a_{21,8} = 0$	$b_{21,8} = 0$	$c_{21,8} = 0$	
$a_{22,1} = 0$	$b_{22,1}$	$c_{22,1}$	
$a_{22,2}$	$b_{22,2}$	$c_{22,2}$	
$a_{22,3}$	$b_{22,3}$	$c_{22,3}$	
$a_{22,4}$	$b_{22,4}$	$c_{22,4}$	
$a_{22,5}$	$b_{22,5}$	$c_{22,5}$	
$a_{22,6}$	$b_{22,6}$	$c_{22,6}$	
$a_{22,7}$	$b_{22,7}$	$c_{22,7} = 0$	
$a_{22,8} = 0$	$b_{22,8} = 1$	$c_{22,8} = 0$	

Fig. 10.1. Data layout for $n = 2$, $N = 7$, $N^* = 8$.

use the same arrangement and since the details can very widely between systems. This particular data layout has been chosen because it makes the odd-even reduction algorithm more efficient for the true STAR model, while the other competitive algorithms (LU factorization, odd-even elimination) are mostly unaffected by the choice of data layout, and can do without extra storage for trivial equations.

Since the STAR forces vector lengths to be $\le$ 64K, we must have $n^2N^* \le 64K = 65{,}536$. For a given value of n, this restricts the system size as follows:

n	2	3	4	5	6	7	8
$64K/n^2$	16384	7281	4096	2621	1826	1337	1024
max N*	16384	4096	4096	2048	1024	1024	1024

These will probably not be serious restrictions in most cases.

A more important restriction on system size is our assumption that the problem and solution will remain core-contained. Assuming a 512K main store, and supposing that temporary storage plus the program amounts to T times the original storage, we obtain the following restrictions on N:

n		2	3	4	5	6	7	8
$\frac{512K}{(T+1)(3n^2+n)}$	T = 1	18724	8738	5041	3276	2279	1702	1310
	T = 2	12483	5825	3360	2184	1533	1134	873
	T = 3	9362	4369	2520	1638	1149	851	655
max N*	T = 1	16384	8192	4096	2048	2048	1024	1024
	T = 2	8192	4096	2048	2048	1024	1024	512
	T = 3	8192	4096	2048	1024	1024	512	512

The amount of temporary storage used depends on the algorithm and the extent to which A and v are overwritten. It can range from practically no temporary storage (other than $O(n^2)$ registers) for the LU factorization to $T \approx m$ for an inefficient implementation of odd-even elimination. Odd-even reduction would typically have $T \leq 2$.

We now consider the costs of data manipulation for the basic algorithms. Major considerations are the speed of the load/store unit on STAR, for register-memory transfer of scalars, and the requirement that vectors be stored as contiguous memory locations.

The LU factorization, executed mostly in scalar mode: The only data manipulation is to load/store between memory and the register file. To a considerable extent this may be overlapped with arithmetic operations. A complete analysis requires cycle counting in an assembly language program and timings from actual runs. Our estimate of execution time (based on an untested cycle count) is that the algorithm will probably be I/O bound. With this in mind, we suggest four possible implementations: (Times given are for $n = 2$, and represent minimal run times assuming complete overlap between scalar arithmetic and load/stores. Actual run time will certainly be greater. For details, see Appendices A, B.)

Version 1, a one-time solution, all scalar mode

$$\text{time} \geq 508N - 362$$

Version 2, factor + solve, partial vector mode

factor, version a, time $\geq 324N + 3421$ (use for $N > \sim 38$)

version b, time $\geq 374N + 1769$

version c, time $\geq 420N - 280$ (use for $N < \sim 38$)

solve, time $\geq 303N + 1039$

The three versions of factorization use increasingly fewer vector operations and increasingly more scalar operations. If more than one system must be solved with the same matrix A, it is best to use Version 2. Our execution time comparison in Section 10.C will use Version 1.

Odd-even elimination, executed in vector mode: An important feature of this algorithm and any data layout like ours based on component vectors is that the conceptual vectors all correspond to the strictest requirements for machine vectors, and hence no additional data manipulation is needed. (cf. [M1] for the tridiagonal case.)

Odd-even reduction, executed in vector mode: If the vector computer model allows vectors to be storage locations in general arithmetic progression, then no additional data manipulation is needed. In a true model of the STAR, however, we need to perform two basic operations: (1) separate a vector into its even and odd components, and (2) rejoin the even and odd parts of a separated vector. The first is accomplished with two STAR compress instructions, the second with a STAR merge instruction. The odd-even separation is needed before each reduction step can be performed, and must be applied to vectors $\underline{a}$, $\underline{b}$, $\underline{c}$, and $\underline{v}$. We illustrate the separation for $\underline{b}$ in fig. 10.2. Having set up a control vector z consisting of $n^2N^*/2$ copies of the bits 10 (a single vector operation on STAR), we execute

$$\text{compress } \underline{b} \rightarrow \underline{b1}(1) \text{ per } z$$

$$\underline{b}(2j-1) \rightarrow \underline{b1}(j), \quad (1 \le j \le \tfrac{1}{2}n^2N^*)$$

$$\text{compress } \underline{b} \rightarrow \underline{b1}(\tfrac{1}{2}n^2N^*) \text{ per } \bar{z}$$

$$\underline{b}(2j) \rightarrow \underline{b1}(\tfrac{1}{2}n^2N^* + j), \quad (1 \le j \le \tfrac{1}{2}n^2N^*).$$

The cost for each compress is $n^2N^* + \frac{1}{8}(\frac{1}{2}n^2N^*) + 92$ and the total cost to

b	z	b1	final state	
$b_{11,1}$	1	$b_{11,1}$	$b_{11,1}^{(1)}$	
$b_{11,2}$	0	$b_{11,3}$	$b_{11,3}^{(1)}$	
$b_{11,3}$	1	$b_{11,5}$	$b_{11,5}^{(1)}$	
$b_{11,4}$	0	$b_{11,7}$	$b_{11,7}^{(1)}$	$b_{ij,k}^{(1)}$, k odd, $1 \le k \le N^*$
$b_{11,5}$	1	$b_{12,1}$	$b_{12,1}^{(1)}$	
$b_{11,6}$	0	$b_{12,3}$	$b_{12,3}^{(1)}$	
⋮	⋮	⋮	⋮	
$b_{12,7}$	1	$b_{22,5}$	$b_{22,5}^{(1)}$	
$b_{12,8}$	0	$b_{22,7}$	$b_{22,7}^{(1)}$	
$b_{21,1}$	1	$b_{11,2}$	$b_{11,1}^{(2)}$	
$b_{21,2}$	0	$b_{11,4}$	$b_{11,3}^{(2)}$	
$b_{21,3}$	1	$b_{11,6}$	$b_{12,1}^{(2)}$	$b_{ij,k}^{(2)}$, k odd, $1 \le k \le N^*/2$
$b_{21,4}$	0	$b_{11,8}$	$b_{12,3}^{(2)}$	
⋮	⋮	⋮	⋮	
$b_{21,8}$	0	$b_{12,8}$	$b_{22,3}^{(2)}$	
$b_{22,1}$	1	$b_{21,2}$	$b_{11,1}^{(3)}$	
$b_{22,2}$	0	$b_{21,4}$	$b_{11,2}^{(3)}$	
$b_{22,3}$	1	$b_{21,6}$	$b_{12,1}^{(3)}$	$b_{ij,k}^{(3)}$, $1 \le k \le N^*/4$
$b_{22,4}$	0	$b_{21,8}$	$b_{12,2}^{(3)}$	
⋮	⋮	⋮	⋮	
$b_{22,8}$	0	$b_{22,8}$	$b_{22,2}^{(3)}$	

Fig. 10.2. Odd-even separation for n = 2, N = 7, N* = 8.

odd-even separate $\underline{a}$, $\underline{b}$, $\underline{c}$, and $\underline{v}$ is $\frac{17}{8}(3n^2 + n)N^* + 736$. Our initial extension of A to N* block equations makes it possible to perform this step using only 8 vector compress operations, independent of n^2. If A had not been extended we would need either $2(3n^2 + n)$ vector operations to compress the vectors one component subvector at a time, or n^2 vector operations to generate a modified control vector consisting of n^2 copies of z(1:N), followed by the original 8 vector compress operations. Lambiotte [L1] rejected use of odd-even reduction for $n > 1$ on this basis.

The net effect of our extended data layout is to reduce the number of vector operations needed to manipulate vectors for the STAR, and thus to reduce vector startup costs. On the other hand, we have increased vector lengths and hence storage from N to N*, and this may be a more important consideration in some cases. The modified control vector approach would then be most natural despite the increased startup costs. In any case, $O(n^3)$ vector operations are needed for the arithmetic part of the reduction step.

So far we have considered only the odd-even separation to prepare for the first reduction. In general, the cost of the separation to prepare for the ith reduction, $1 \le i \le m - 1$, is $\frac{17}{8}(3n^2 + n)N_i^* + 736$, $N_i^* = 2^{m+1-i}$, and the total cost for m-1 reductions is $\frac{17}{4}(3n^2 + n)(N^* - 2) + 736(m-1)$. For the case $n = 2$ this yields $59.5N^* + 736m - 855$. We will see shortly that this is roughly half as much time as is spent in the arithmetic operations, so the overhead of dealing with a restrictive machine vector definition can be considerable.

Data manipulation for the back substitution is simplified by saving the odd-indexed equations of $A^{(i)}$ in appropriate form for vector operations (see fig. 10.2 for the final state of $\underline{b}$). Having computed $x^{(i+1)}$ = the even components of $x^{(i)}$, we compute the odd components of $x^{(i)}$ and merge the two sets into $x^{(i)}$. The merge operation on STAR is comparatively more expensive than the compress operation, but we are only dealing with the solution vector and not blocks of A. The cost of the merge operation at stage i is $3nN_i^* + 123$, and the total cost for m-1 reductions is $6n(N^* - 2) + 123(m - 1)$. For n = 2 this yields $12N^* + 123m - 147$.

p-fold reduction: Suitable data arrangement and use of control vectors for p-fold reduction can readily be generalized from our discussion of odd-even reduction. The odd-even separation generalizes to separation into p vectors, which requires p compress instructions per storage vector; the cost per reduction would be $(p + \frac{1}{8})(3n^2 + n)N_i^* + 368p$, $N_i^* = p^{m+1-i}$, $m = \lceil \log_p N \rceil$. The back substitution requires merging p vectors into 1, using p-1 merge instructions at cost of $3(\frac{p+1}{2} - \frac{1}{p})nN_i^* + 123(p-1)$ to go from $x^{(i+1)}$ to $x^{(i)}$. It is clear that these costs are minimized by $p = 2$.

Jacobi iterations: Data manipulation is unnecessary when A and v or B[A] and $D[A]^{-1}v$ are stored according to our component subvector scheme.

10.C. Comparative Timing Analysis.

We close with an execution time comparison of algorithms for general block tridiagonal systems with 2×2 blocks, based on timing information for the CDC STAR-100. This will expand the crude comparison of the LU factorization and odd-even reduction given in Section 4.D, and illustrates many earlier statements concerning the cost of various methods. Essential considerations include vector startup costs, creating a substantial penalty for operations on short vectors, and data manipulation to deal with restrictive definitions of vector operands. Section 10.B began our discussion of the latter problem.

Our initial simplified analysis is based on the following conventions, which allow us to consider only arithmetic costs:

1. we assume
   a. vector operands may be any sequence of storage locations in arithmetic progression,
   b. pivoting is never necessary;
2. we ignore
   a. instruction overlap capabilities,
   b. address calculations, loop control and termination tests,
   c. load/store costs for scalar operands,
   d. storage management costs, as a consequence of assumption 1.a.

Our second, and more realistic analysis will demand that vector operands consist of consecutive storage locations, so we must consider storage management as explained in Section 10.B. Overlap, loop control and scalar load/store will also be considered, but not address calculation or termination of iterations. Algorithms and timing derivations are given in Appendix B; here we present the main results and make comparisons.

Execution times for the basic direct methods are initially estimated as follows ($m = \lceil \log_2 N \rceil$):

1. the block LU factorization, using scalar operations only (LU):
   $1012N - 784$
2. odd-even elimination, using vector operations only (OEE):
   $94.5Nm + 12999m - 93.5N - 1901$

3. odd-even reduction, using vector operations except for the final block system (OER): $106.5N + 14839m - 14717.5$

4. p-fold reduction, $p \geq 3$:

$$145N + 19206m \left(\frac{p-1}{\log_2 p}\right) - (19351p - 19579)$$

We can first dispose of p-fold reduction as a competitive method for general problems since either LU or OER is always cheaper. Execution time ratios show the benefits of using OER in preference to LU and OEE.

	$N = 1023$	$N = 8191$	limit, $N \to \infty$
OER: LU	.23	.13	.11
OER: OEE	.24	.11	0.

The importance of vector startups is seen by comparing OER and LU, for the latter is faster when $N < 100$. OER works by successively reducing vector lengths, and at some point this becomes inefficient. A polyalgorithm combining the two methods is straightforward: if $m \leq 6$ then use LU, otherwise do m-5 reductions, solve the remaining system by LU, and back substitute. This reduces the time for OER to $106.5\,N + 14839m - 46908.5$, and for $N = 1023$ it represents a savings of 13%.

We also note that vector startups are the dominant term for OER when $N \leq 1532$, and are the dominant term for OEE when $N \leq 137$. Of course, OEE uses $O(N \log N)$ operations vs. $O(N)$ for the other methods and soon becomes inefficient. Nevertheless, it is faster than LU for $455 \leq N \leq 1023$, although it is always slower than OER.

Our conclusion is that the OER - LU polyalgorithm is the best direct method of those considered here.

Now consider the cost of individual eliminations or reductions with respect to the total cost of the algorithm. For odd-even elimination, each elimination step has roughly equal cost, the last being slightly cheaper. Thus each step costs about 1/m of the total cost. For odd-even reduction, the cost per reduction decreases as the algorithm proceeds:

cost of the kth reduction as a fraction of total cost

	k = 1	2	3	4	5	6	total cost
N = 1023, m = 10	28%	17%	12%	8%	7%	7%	242,622
N = 8191, m = 13	43%	22%	12%	7%	4%	3%	1,050,531

(cost for kth red. = $106.5(2^{m-k}) + 14839$ when $N = 2^m - 1$)

For small N, when startup costs are dominant, reduction costs are approximately equal, while for large N, when true arithmetic costs are dominant, the kth reduction costs about $1/2^k$ of the whole. This has immediate consequences for the effectiveness of incomplete reduction, since the omitted reductions are only a small part of the total cost when N is large.

For our comparison of semidirect and iterative methods, the goal will be to approximate x with a relative error of 1/N, or to reduce the initial error by a factor of 1/N. Thus if we perform either k odd-even eliminations or k reductions, we want $\|B[H_{k+1}]\| \le 1/N$ or $\|B[A^{(k+1)}]\| \le 1/N$, assuming $\|B[A]\| < 1$. Theorem 7.1 is then applied to produce the desired approximation by incomplete elimination or reduction. Taking $\beta = \|B[A]\|$, we have $\|B[H_{k+1}]\|$, $\|B[A^{(k+1)}]\| \le \beta^{2^k}$, and demanding $\beta^{2^k} \le 1/N$ yields

$$k \ge \log_2((\log_2 N) / (-\log_2 \beta)).$$

The execution time for k eliminations is

$$94.5Nk + 12999k + 10.5N - 105(2^k) + 1205,$$

and the time for k reductions is

$$106.5N + 14839k - 96(2^{m-k}) + 1287.$$

Typical savings for incomplete reduction over complete reduction would be:

$\beta = .9$, pick $k \geq \log_2 \log_2 N + 2.718$

$N = 1023$, $k = 7$, savings = 12%

$N = 8191$, $k = 7$, savings = 7%

$\beta = \frac{2}{3}$, pick $k \geq \log_2 \log_2 N + .774$

$N = 1023$, $k = 5$, savings = 25%

$N = 8191$, $k = 5$, savings = 12%

$\beta = \frac{1}{2}$, pick $k \geq \log_2 \log_2 N$

$N = 1023$, $k = 4$, savings = 33%

$N = 8191$, $k = 4$, savings = 16%

Comparison to the OER - LU polyalgorithm decreases the savings.

We next consider iterative methods and their combination with semi-direct methods. One step of the block Jacobi iteration costs $22.5N + 3031$, while one step of the block Jacobi semi-iteration costs $26.5N + 3408$ plus overhead to estimate parameters such as $\rho = \rho(B[A])$. Execution time may be reduced to $12N + 1840$ ($16N + 2217$) per iteration by directly computing $B[A]$ and $D[A]^{-1}v$ (cost = $38.5N + 4367 \doteq 3$ iterations).

We will use the Jacobi iteration in its simplest form when only a few steps are demanded, and Jacobi semi-iteration when many steps are needed. The number of iterations needed by J-SI to reduce the initial error by a factor of $1/N$ are estimated to be

The LU factorization is always faster than OEE, and is faster than OER for $N \le 312$. Vector startup costs are dominant in OEE for $N \le 152$, and in OER for $N \le 798$. The OER-LU polyalgorithm should perform m-6 reductions, solve the remaining system by LU, and back substitute. The time is $183N + 16233m - 70677$, yielding a 16% saving over OER for N = 1023 but only a 3% saving for N = 8191.

The execution time for odd-even elimination is allocated as follows:

	N = 1023	N = 8191
arithmetic	96%	96%
overhead	4%	4%
results	87%	98%
vector startups	13%	2%

Analysis of the total time for odd-even reduction shows the greater importance of overhead operations:

N = 1023	results	startups	
arithmetic	33%	40%	73%
overhead	23%	4%	27%
	56%	44%	

N = 8191	results	startups	
arithmetic	52%	11%	62%
overhead	37%	1%	38%
	88%	12%	

limit, $N \to \infty$	results	startups	
arithmetic	58%	-	58%
overhead	42%	-	42%
	100%	-	

The increase in time for OER due to overhead is 37% for N = 1023, 61% for N = 8191, and 72% in the limit as $N \to \infty$. As a consequence of the data layout of Section 10.B, overhead startup costs are quite small, but the overhead result costs cannot be ignored and probably cannot be decreased. The strong requirement that machine vectors consist of contiguous storage locations leads to much "wasted" effort in execution time.

Most of the execution time of OER is actually spent in the arithmetic and vector compress operations:

cost of individual operation as fraction of total cost

	N = 1023	N = 8191	limit, $N \to \infty$
divide	12%	12%	12%
multiply	43%	35%	32%
add/subtract	18%	15%	14%
compress	20%	29%	33%
merge	4%	6%	7%
transmit	3%	3%	3%

The compress operation assumes more importance as N increases, again reflecting its low startup costs.

As noted earlier, most of OER's time is spent in the first few reductions and subsequent back substitutions.

$$\rho = .9, \quad 1.5m \text{ iterations},$$
$$\rho = \tfrac{2}{3}, \quad .75m \text{ iterations},$$
$$\rho = \tfrac{1}{2}, \quad .5m \text{ iterations}.$$

For $\rho = \frac{2}{3}$, for example, J-SI is never faster than OER.

As suggested in Section 7.A, we can combine incomplete elimination or reduction with Jacobi iterations. If k steps of elimination or reduction are used, followed by ℓ iterations and back substitution, k and ℓ should be chosen so that $\beta^{2^k \ell} \le 1/N$, or

$$(*) \qquad k + \log_2 \ell \ge \log_2((\log_2 N) \;/\; (-\log_2 \beta)).$$

The time for the OER-J method would then be

$$106.5N + 14839k - 96(2^{m-k}) + 1287 + \ell(19(2^{m-k}) + 2615),$$

and it only remains to minimize the time subject to (*). For $\beta = \frac{2}{3}$, $N = 1023$, the resulting values are $k = 2$, $\ell = 5$, which represent a savings of 37% over OER. For $\beta = \frac{2}{3}$, $N = 8191$, the resulting values are $k = 3$, $\ell = 3$, which represent a savings of 15% over OER.

Of course, to determine k and ℓ we need to know β. While it is true that $\beta = \|B[A]\|$ can be computed directly from A at a cost of $46.5N + 3881$, or from B[A] at a cost of $15N + 308$, we recognize that half the rows of B[A] are available as part of the first odd-even reduction. An estimate of $\|B[A]\|$ can be computed from these rows at a cost of $7.5N + 308$. For large N even this cost considerably reduces any benefits gained from using incomplete reduction, and points out the importance of an analytic estimate of $\|B[A]\|$ rather than a computational estimate.

This ends our initial comparison of algorithms. We now make the assumption that vector operands are consecutive storage locations, allow overlap of scalar operations, and consider the cost of loop control and scalar load/store operations. The net effect is to lower the time estimate for the LU factorization and raise the time estimate for odd-even reduction. Odd-even elimination remains substantially unchanged. Rather than simply recompute the figures presented earlier, which will be done in a few cases, we try to indicate where the algorithms spend most of their time.

Revised time estimates are given below. These should give a better indication of actual runtime on STAR, but are probably all underestimates; we note changes from the first set of timings. $m = \lceil \log_2 N \rceil$.

1. LU factorization, (LU) $600N - 150$

   (a 40% decrease due to use of instruction overlap)

2. Odd-even elimination, (OEE) $98.5Nm + 13403m - 101.5N - 2261$

   (a 4% increase due to overhead costs)

3. Odd-even reduction. (OER) $183N + 16233m - 16108$

   (a 10-70% increase due to overhead costs; larger N incurs larger increases).

Time ratios show the effect of decreasing the LU time and increasing the OER time

	$N = 1023$	$N = 8191$	limit, $N \to \infty$
OER:LU	.54	.34	.31
OER:OEE	.32	.17	0.

cost of ith reduction as fraction of total cost

i =	1	2	3	4	5	6
N = 1023	33%	19%	12%	8%	7%	6%
N = 8191	45%	23%	12%	6%	4%	2%
limit, N → ∞	50%	25%	13%	6%	3%	2%

Of the total time, 86% is spent in the reduction phase and 14% in the back substitution.

One of the effects of the OER-LU polyalgorithm is to reduce the importance of vector startup costs, as seen below.

	N = 1023	N = 8191
scalar time	14%	2%
vector time	86%	98%
results	63%	91%
startups	23%	7%

Of course, when startup costs are already small not much improvement can be made. Incomplete reduction and incomplete reduction + Jacobi iterations display similar results. ($\beta = \| B[A] \|$).

Incomplete reduction, savings over OER

	N = 1023		N = 8191	
	k	savings	k	savings
$\beta = .9$	7	10%	7	5%
$\frac{2}{3}$	5	21%	5	9%
$\frac{1}{2}$	4	27%	4	13%

Incomplete reduction + Jacobi iteration, saving over OER

	N = 1023			N = 8191		
	k	ℓ	savings	k	ℓ	savings
$\beta = .9$	3	9	22%	4	6	8%
$\frac{2}{3}$	2	5	36%	2	6	13%
$\frac{1}{2}$	1	5	47%	2	4	21%

As seen earlier, the semidirect methods are best applied when the omitted odd-even reductions make up a significant part of the OER time. There is considerable advantage to using incomplete reduction plus Jacobi iterations, roughly doubling the savings obtained by simple incomplete reduction.

APPENDIX A

VECTOR COMPUTER INSTRUCTIONS

This section describes those features of the CDC STAR-100 needed for our comparison of algorithms in Section 10. Information given here is derived from CDC reference manuals and timing measurements made at Lawrence Livermore Laboratory [C2], and by no means represents a complete discussion of the STAR's facilities.

Execution times are for full-word (64 bit) floating point instructions, given as the number of 40 nanosecond cycles needed for each instruction. All times should be accurate to within 5% except as noted, though actual performance may be degraded somewhat due to memory conflicts.

A vector on STAR consists of N consecutive storage locations, $1 \le N \le 65,536$. The vector length restriction is usually not serious, but the need for consecutive storage is crucial. Other vector computers, the TI ASC in particular, do not have this restriction. Both machines are pipeline computers with the usual scalar instructions as well as vector instructions.

Useful features of vector instructions on STAR include:

broadcast constants - Most vector instructions allow a scalar (the broadcast constant, held in a register) to be transmitted repeatedly to form one or both of the vector operands. Use of this facility decreases execution time in a few cases.

control vectors - Bit vectors can modify the effect of vector instructions, such as suppressing storage to selected components of the result vector or determining which operand components are to be used in an operation.

Use in vector data manipulation (compress and merge) is explained later.

offset - Modification to the base address of operands.

sign control - Certain vector operations allow the sign of the operands to be modified before execution.

Individual classes of instructions follow.

scalar (register) instructions. Issue time is the number of cycles required to initiate execution. Shortstop time is the time after issue when the result is first available, but not yet placed in a register. The result must be used at the precise time given; if not, the result cannot be used until it is placed in a register. Result-to-register time is the number of cycles after issue when the result is placed in a register. Overlapped execution is possible, but the exact effect on execution time is best determined by actual testing, which we have not done.

STAR instruction	operation	issue	shortstop	result to register
62	add, normalized	2	6	11
66	subtract, normalized	2	6	11
6B	multiply, significant	2	10	15
6F	divide, significant	3	-	43
73	square root	3	-	71
78	register transmit	2	4	9

scalar (register) load/store instructions. Up to three load/store instructions may be stacked in the load/store unit at any given time. The stack is processed sequentially at a rate of 19 cycles per load and 16 cycles per store minimum, assuming no memory conflicts. Once a memory bank has been referenced it remains busy for 31 cycles; during this time further references to the bank are delayed and the conflict will affect references to other memory banks. Start to finish for a single load requires at least 31 cycles, and a minimum of 66 cycles are required to store an item and reload it.

STAR instruction	operation	issue	result to register	load/store unit busy
7E	load	3	28	19
7F	store	3	-	16

branch instructions. Execution time for a branch can range from 8 to 34 cycles, depending on the particular instruction, whether a branch is actually made, whether the target instruction is in the instruction stack, etc. We will be only slightly pessimistic if we take 40 cycles to be the time for a typical branch.

vector operation instructions. Execution time consists of an initial startup time followed by sequential appearance of result vector components, or startup time followed by processing time of operands if a scalar result is generated. Vector instructions may not be overlapped except for a negligible part of the startup time. In the table below, N is the length of the input vector(s) and V is the number of omitted items due to use of a control vector. Times for instructions D8 - DC may vary as much as $\pm 20\%$.

STAR instruction	vector operation	time	effect
82	add, normalized	$\frac{1}{2}N + 71$	$c_i \leftarrow a_i + b_i$
86	subtract, normalized	$\frac{1}{2}N + 71$	$c_i \leftarrow a_i - b_i$
8B	multiply, significant	$N + 159$	$c_i \leftarrow a_i \times b_i$
8F	divide, significant	$2N + 167$	$c_i \leftarrow a_i \div b_i$
93	square root	$2N + 155$	$c_i \leftarrow \sqrt{a_i}$
98	transmit	$\frac{1}{2}N + 91$	$c_i \leftarrow a_i$
99	absolute value	$\frac{1}{2}N + 91$	$c_i \leftarrow \lvert a_i \rvert$
D8	maximum	$6N - 2V + 95$	$c \leftarrow \max a_i$
DA	summation	$6.5N - 2V + 122$	$c \leftarrow \sum_{i=1}^{N} a_i$
DB	product	$6N - V + 118$	$c \leftarrow \prod_{i=1}^{N} a_i$
DC	inner product	$6N - V + 130$	$c \leftarrow \sum_{i=1}^{N} a_i b_i$

<u>vector data manipulation instructions</u>. The compress instruction transmits selected components of one vector into a shorter result vector. Given a vector $\underline{a}$ and a control vector $\underline{z}$, the instruction essentially executes the code

```
j ← 1
for i = 1, 2, ..., N
    if $z_i = 1$ then {$c_j \leftarrow a_i$; j ← j+1}.
```

The result vector $\underline{c}$ has length bitsum($\underline{z}$). The merge instruction "shuffles" two vectors into a third, according to the control vector:

```
length($\underline{c}$) = length($\underline{z}$) = length($\underline{a}$) + length($\underline{b}$)
j ← 1; k ← 1
for i = 1, 2, ..., length($\underline{c}$)
    if $z_i$ = 1 then {$c_i$ ← $a_j$ ; j ← j+1}
              else {$c_i$ ← $b_k$ ; k ← k+1}
```

In the table below, M_1 = length($\underline{a}$), M_2 = length($\underline{c}$), R = number of input items broadcast from register.

STAR instruction	operation	time
BC	compress $\underline{a} \rightarrow \underline{c}$ per $\underline{z}$	$M_1 + \frac{1}{8}M_2 + 92$
BD	merge $\underline{a},\underline{b} \rightarrow \underline{c}$ per $\underline{z}$	$3M_2 - 2R + 123$

Execution times for these instructions may vary as much as $\pm 20\%$.

APPENDIX B

SUMMARY OF ALGORITHMS AND TIMINGS

We collect the algorithms discussed in the main text, and present timing formulae in general terms for a vector computer. This is then specialized to block tridiagonal systems with 2 × 2 blocks using timing information for the CDC STAR-100. A comparison of methods based on the latter set of timings is given in Section 10.C. It must be emphasized that our time estimates have not been verified by direct testing on STAR.

The algorithms solve $Ax = v$, $A = (a_j, b_j, c_j)_N$, with blocks of uniform size $n \times n$. The basic arithmetic operations are: (S = add or subtract, M = multiply, D = divide)

1. factor an $n \times n$ block b

   cost $= \frac{1}{2}(n^2 - n)D + \frac{1}{6}(2n^3 - 3n^2 + n)(M + S)$
2. having factored b, solve $bg = f$, where f is $n \times n'$

   cost $= n'(nD + (n^2 - n)(M + S))$
3. product of $n \times n$ and $n \times n'$ blocks

   cost $= n'(n^2M + (n^2 - n)S)$
4. difference of two $n \times n'$ blocks

   cost $= n'(nS)$

Depending on context, the symbols S, M, D can mean either a single scalar operation or a vector operation with vector length specified by the algorithm.

For each algorithm we give a code in terms of blocks of A, since the translation to individual scalar or vector operations is generally obvious.

In those cases where additional detail seems to be necessary it is provided. Two basic sets of timings are given, five formulae in all for each algorithm:

1. counting arithemtic operations only, assuming machine vectors can be any sequence of storage locations in arithmetic progression, and ignoring pivoting, instruction overlap, address calculation, loop control, termination tests for iterations, load/store costs for scalar operands, and any other forms of storage management
   - (a) for systems with $n \times n$ blocks in general terms for a vector computer
   - (b) for systems with $n \times n$ blocks, with timing data for the CDC STAR-100
   - (c) for systems with 2×2 blocks, with STAR data
2. a more realistic model of STAR, requiring machine vectors to be contiguous storage locations, allowing overlap of scalar operations, counting storage management and load/store costs, and loop control to some extent, but still ignoring pivoting, address calculations and termination tests for iterations
   - (a) for systems with $n \times n$ blocks, with STAR data
   - (b) for systems with 2×2 blocks, with STAR data.

Additional notes are provided in many cases. Timings 1(c) and 2(b) are used in Section 10.C.

A few remarks on the timings are necessary. Since we eventually want estimates for 2×2 blocks, the only loop control that will be counted is that on the main loops. Operations on n $\times$ n matrices or n-vectors are assumed to be expanded into straightline code.

The address calculations that have been ignored here would be executed in scalar mode, and probably can be completely overlapped with scalar arithmetic operations in the LU factorization. Their effect on the odd-even methods would be to increase the $O(m)$ terms, $m = \lceil \log_2 N \rceil$, but this probably represents only a small relative increase since the $O(m)$ terms consist of vector startup costs and are already large. While scalar operations can sometimes destroy the efficiency of a vector code, we believe that our basic conclusions will not be altered by including address calculations.

1. Block LU factorization, using scalar operations mostly

Algorithm:

```
$d_1 = b_1$ ; $f_1 = v_1$
for $i = 2, \dots, N$
    solve $d_{i-1}(u_{i-1}\ g_{i-1}) = (c_{i-1}\ f_{i-1})$
    $d_i = b_i - a_i u_{i-1}$
    $f_i = v_i - a_i g_{i-1}$
solve $d_N x_N = f_N$
for $i = N-1, \dots, 1$
    $x_i = g_i - u_i x_{i+1}$
```

Storage handling:

If desired we can overwrite $d_i \to b_i$, $\ell_i = a_i d_{i-1}^{-1} \to a_i$, $u_i \to c_i$, $x_i \to g_i \to f_i \to v_i$. The factorization loop needs to save either d_i and f_i or u_i and g_i, a total of $n^2 + n$ stores and subsequent loads. If d_i, f_i are saved then the second loop would be

```
for $i = N-1, \dots, 1$
    solve $d_i x_i = f_i - c_i x_{i+1}$
```

with a corresponding increase in execution time.

Variations of the basic algorithm, with estimate of minimum runtime due to load/store unit:

Version 1, a one-time solution.

```
initialization loads $b_1$, $v_1$
factorization loop loads $c_{i-1}$, $a_i$, $b_i$, $v_i$
                   stores $u_{i-1}$, $g_{i-1}$
initialization stores $x_N$
```

back substitution loop loads u_i, g_i

stores x_i

At 19 cycles per load and 16 per store,

$$\text{total time} \geq \{19(4n^2 + 2n) + 16(n^2 + 2n)\}(N-1)$$
$$+ 19(n^2 + n) + 16(n)$$
$$= (92n^2 + 70n)(N-1) + 16n^2 + 35n$$

for $n = 2$, total time $\geq 508N - 362$

Version 2, factor + solve, $A = (\ell_j, I, 0)(0, d_j, 0)(0, I, u_j)$

Factorization requires $n^2(3N - 2)$ loads for A.

Version a. factorization loop stores d_i only, and computes

$\ell_j = a_j d_{j-1}^{-1}$, $u_j = d_j^{-1} c_j$ by vector operations, vector length = N.

$$\text{total time} \geq 19n^2(3N - 2) + 16n^2N + \frac{1}{2}(5n^2 - n)D + \frac{1}{6}(14n^3 - 15n^2 + n)(M+S),$$

$D = \tau_{\div}N + \sigma_{\div}$, etc.

for $n = 2$, total time $\geq 323.5N + 3421$

Version b. factorization loop stores d_i, u_i, computes ℓ_i by vector operations, vector length = N.

$$\text{total time} \geq 19n^2(3N - 2) + 16n^2(2N - 1)$$
$$+ \frac{1}{2}(3n^2 - n)D + \frac{1}{6}(8n^3 - 9n^2 + n)(M+S)$$

for $n = 2$, total time $\geq 373.5N + 1769$

Version c. factorization loop stores d_i, u_i, ℓ_i.

$$\text{total time} \geq 19n^2(3N - 2) + 16n^2(3N - 2)$$

for $n = 2$, total time $\geq 420N - 280$

Solution of $Ax = v$, given the factorization of A.

forward elimination loads v_1 initially

loop loads ℓ_i, v_i, stores f_i

solve $d_j g_j = f_j$ with vector operations.

back substitution loads x_N initially

loop loads u_i, g_i, stores x_i

total time $\geq 19(2n^2(N-1) + 2nN) + 16(2nN)$

$+ \frac{1}{2}(n^2 + n)D + \frac{1}{6}(2n^3 + 3n^2 - 5n)(M+S)$

for $n = 2$, total time $\geq 302.5N + 1039$

Timings:

1. Version 1, for initial comparison, arithmetic operations without overlap, load/stores ignored.

   (a) $\{\frac{1}{2}(3n^2 + n)t_D + \frac{1}{6}(14n^3 + 9n^2 - 5n)(t_M + t_S)\}N$
   $- \{n^2 t_D + (2n^3 + n^2)(t_M + t_S)\}$

   (b) with STAR data: $(70n^3 + 114n^2 - 2n)N - (60n^3 + 76n^2)$

   (c) for $n = 2$, with STAR data: $1012N - 784$

2. Version 1, for second comparison, allowing overlap and including load/stores.

   In the absence of facilities for direct testing on STAR, a scalar code was written for the special case $n = 2$ and time estimates computed from that code. The details are tedious and well-illustrated by the simpler tridiagonal code in [L2], so they will not be given here. The main time estimates are

initialize - loads + loop set up	200 cycles
factorization loop	400(N-1)
initialize	250
back substitution loop	200(N-1)
total time estimated to be	600N - 150 cycles

   This timing is probably optimistic due to unforeseen memory and overlap conflicts.

Notes:

1. Gaussian elimination, taking advantage of the block structure, using scalar operations, requires the same execution time as the block LU factorization.

2. If u_j, $1 \le j \le N-1$, is saved in the factorization, then $\| B[A_N] \|$ may be computed following the first loop at a cost of

$$((n-1)\tau_+ + \tau_{max})(nN) + (n-1)\sigma_+ + \sigma_{max}$$

using the vector maximum instruction. This yields $13N + 166$ for $n = 2$ on STAR.

2. Odd-even elimination, using vector operations

We give the algorithm and timing for $N = 2^m - 1$, deriving formulae solely in terms of N and m. Timing for other values of N is approximated by setting $m = \lceil \log_2 N \rceil$.

```
Algorithm: $a_j^{(1)} = a_j$, etc.
    for $i = 1,2,\ldots,m$  $(r = 2^{i-1})$                                          vector
                                                                                    lengths
        for each $j = 1,2,\ldots,N$
            solve $b_j^{(i)}(\alpha_j\ \gamma_j\ \varphi_j) = (a_j^{(i)}\ c_j^{(i)}\ v_j^{(i)})$                N, N-r
            take $\alpha_j = \gamma_j = 0$, $\varphi_j = 0$ if $j < 1$ or $j > N$
            $b_j^{(i+1)} = b_j^{(i)} - a_j^{(i)}\gamma_{j-r} - c_j^{(i)}\alpha_{j+r}$                      N-r
            $v_j^{(i+1)} = v_j^{(i)} - a_j^{(i)}\varphi_{j-r} - c_j^{(i)}\varphi_{j+r}$                      N-r
            $a_j^{(i+1)} = -a_j^{(i)}\alpha_{j-r}$                                         N-2r
            $c_j^{(i+1)} = -c_j^{(i)}\gamma_{j+r}$                                         N-2r
    for each $j = 1,2,\ldots,N$
        solve $b_j^{(m+1)} x_j = v_j^{(m+1)}$
```

Storage handling: temporary vectors are needed for the factorization of $b_j^{(i)}$ and for α_j, γ_j, φ_j computed at each stage, and we can overwrite $b_j^{(i+1)} \to b_j^{(i)}$, $a_j^{(i+1)} \to a_j^{(i)}$, etc.

Timing:

1. arithmetic only

$$\text{(a)}\quad \{\tfrac{1}{2}(5n^2 + n)\tau_D + \tfrac{1}{6}(38n^3 + 3n^2 - 5n)\tau_M + \tfrac{1}{6}(38n^3 - 9n^2 - 5n)\tau_S\}Nm + \{\tfrac{1}{2}(5n^2 + n)\sigma_D + \tfrac{1}{6}(38n^3 + 3n^2 - 5n)\sigma_M + \tfrac{1}{6}(38n^3 - 9n^2 - 5n)\sigma_S)\}\, m$$

$$- \{\tfrac{1}{2}(3n^2 - n)\tau_D + \tfrac{1}{6}(46n^3 - 3n^2 + 5n)\tau_M + \tfrac{1}{6}(46n^3 - 27n^2 + 5n)\tau_S\} N$$
$$+ \{(2n^3)\tau_M + (2n^3 - 2n^2)\tau_S + \tfrac{1}{2}(n^2 + n)\sigma_D + \tfrac{1}{6}(-10n^3 + 3n^2 - 5n)\sigma_M + \tfrac{1}{6}(-10n^3 + 15n^2 - 5n)\sigma_S\}$$

(b) with STAR data,

$$\tfrac{1}{4}(38n^3 + 19n^2 - n)Nm + \tfrac{1}{6}(8740n^3 + 2343n^2 - 649n)m - \tfrac{1}{4}(46n^3 + n^2 + n)N - \tfrac{1}{6}(2282n^3 - 2037n^2 + 649n)$$

(c) for $n = 2$, with STAR data,

$$94.5\ Nm + 12999m - 93.5N - 1901.$$

2. With overwriting of $a_j^{(i+1)} \to a_j^{(i)}$, $c_j^{(i+1)} \to c_j^{(i)}$, it is necessary to develop the products one row at a time in a temporary vector, and then use a transmit operation on the whole row. For specifics see the discussion of odd-even reduction to follow. The added cost for loop control and transmit instructions is

$$40m + \sum_{i=1}^{m-1} 2n(\tfrac{1}{2}n(N-2^i) + 91) = 40m + n^2(N(m-2) + 1) + 182n(m-1),$$

or $4Nm + 404m - 8N - 360$ for $n = 2$.

(a) with STAR data, total time is

$$\tfrac{1}{4}(38n^3 + 23n^2 - n)Nm + \tfrac{1}{6}(8740n^3 + 2343n^2 + 443n + 240)m - \tfrac{1}{4}(46n^3 + 9n^2 + n)N - \tfrac{1}{6}(2282n^3 - 2043n^2 + 1741n)$$

(b) for $n = 2$, with STAR data, total time is

$$98.5Nm + 13403m - 101.5N - 2261.$$

Notes:

1. $\|B[A]\|$ may be computed in the first time through the loop at a cost of $((2n-1)\tau_+ + \tau_{max})(nN) + ((2n-1)\sigma_+ + \sigma_{max})$. This yields $15N + 308$ for $n = 2$ on STAR. $\|B[H_i]\|$ may be computed at a cost of $15(N - 2^{m-i}) + 308$ at the appropriate time through the loop.

3. Odd-even reduction, using vector operations

As with odd-even elimination, the algorithm and timings are derived for $N = 2^m - 1$. For general N take $m = \lceil \log_2 N \rceil$ in the timing formulae, even though $m = \lceil \log_2 N+1 \rceil$ is the proper choice. The algorithm is given in detailed form, using the data layout and storage handling described in Section 10.B. For the basic form, see Section 4.D.

```
Algorithm: $A^{(1)} = A$, $w^{(1)} = v$, $N^* = 2^m$
    set up control vector $z = (1\ 0\ 1\ 0\ \ldots)$, length $n^2 N^*$ bits
        cost not counted in timing, but = one instruction on STAR
    for $i = 1, 2, \ldots, m-1$ ($N_i = 2^{m+1-i} - 1$, $N_i{}^* = 2^{m+1-i}$)
        odd-even separate $A^{(i)}$, $w^{(i)}$
            8 compress operations, cost = $\frac{17}{8}(3n^2 + n)N_i{}^* + 736$
        factor $b_{2j+1}{}^{(i)}$, $(0 \le j \le N_{i+1})$
            factorization overwrites $b_{2j+1}{}^{(i)}$
            need a temporary vector, length $N_{i+1}{}^*$
            cost = $\frac{1}{2}(n^2 - n)D + \frac{1}{6}(2n^3 - 3n^2 + n)(M + S)$,
                vector lengths = $N_{i+1}{}^*$.
        solve $b_{2j+1}{}^{(i)} (\alpha_{2j+1}{}^{(i)}\ \gamma_{2j+1}{}^{(i)}\ \varphi_{2j+1}{}^{(i)})$
                $= (a_{2j+1}{}^{(i)}\ c_{2j+1}{}^{(i)}\ w_{2j+1}{}^{(i)})$, $(0 \le j \le N_{i+1})$
            $\alpha_{2j+1}{}^{(i)}$ overwrites $a_{2j+1}{}^{(i)}$, etc.
            need a temporary vector, length $N_{i+1}{}^*$
            cost = $(2n + 1)(nD + (n^2 - n)(M + S))$,
                vector lengths = $N_{i+1}{}^*$
        $b_j{}^{(i+1)} = b_{2j}{}^{(i)} - a_{2j}{}^{(i)}\ \gamma_{2j-1}{}^{(i)} - c_{2j}{}^{(i)}\ \alpha_{2j+1}{}^{(i)}$, $(1 \le j \le N_{i+1})$
            overwrites $b_{2j}{}^{(i)}$
            need a temporary vector, length $N_{i+1}$
        cost = $2n^3(M + S)$
          for simplicity, take vector lengths = $N_{i+1}*$
    $w_j^{(i+1)} = w_{2j}^{(i)} - a_{2j}^{(i)}\varphi_{2j-1}^{(i)} - c_{2j}^{(i)}\varphi_{2j+1}^{(i)}$,
        overwrites $w_{2j}^{(i)}$
        need a temporary vector, length $N_{i+1}$
        cost = $2n^2(M + S)$
          for simplicity, take vector lengths = $N_{i+1}*$
    $a_j^{(i+1)} = -a_{2j}^{(i)}\alpha_{2j-1}^{(i)}$, $(2 \le j \le N_{i+1})$
        overwrites $a_{2j}^{(i)}$, product developed one row at a time:
        for k = 1, ..., n
          for $\ell$ = 1, ..., n
            $t_{\ell j} = a_{k1,2j}^{(i)}\alpha_{1\ell,2j-1}^{(i)}$, $(1 \le j \le N_{i+1})$
            for r = 2, ..., n
              $t_{\ell j} = t_{\ell j} + a_{kr,2j}^{(i)}\alpha_{r\ell,2j-1}^{(i)}$, $(1 \le j \le N_{i+1})$
            $a_{k\ell,j} = -t_{\ell,j}$, $(1 \le \ell \le n; 1 \le j \le N_{i+1}*)$
        cost = $n^3M + (n^3 - n^2)S$, vector length = $N_{i+1}$,
          + n transmits, vector length = $nN_{i+1}*$.
        need two temporary vectors, length = $nN_{i+1}$ and $N_{i+1}$.
    $c_j^{(i+1)} = -c_{2j}^{(i)}\gamma_{2j+1}^{(i)}$, $(1 \le j \le N_{i+1} - 1)$
          as for $a_j^{(i+1)}$

solve $b_1^{(m)} x_1^{(m)} = w_1^{(m)}$, scalar mode
    cost = $\frac{1}{2}(n^2 + n)D + \frac{1}{6}(2n^3 + 3n^2 - 5n)(M + S)$

for i = m-1, ..., 1
    $x_{2j+1}^{(i)} = \varphi_{2j+1}^{(i)} - \alpha_{2j+1}^{(i)} x_j^{(i+1)} - \gamma_{2j+1}^{(i)} x_{j+1}^{(i+1)}$, $(0 \le j \le N_{i+1})$
        overwrites $\varphi_{2j+1}^{(i)}$
        cost = $2n^2(M + S)$, vector length = $N_{i+1}*$
```

$x_j^{(i+1)} \rightarrow$ temporary

cost = transmit, vector length = nN_{i+1}^*

merge $x_{2j+1}^{(i)}$, temporary $\rightarrow x^{(i)}$

cost = $3nN_i^* + 123$

Timings:

1. arithmetic only,

(a) $$\{\frac{1}{2}(5n^2 + n)\tau_D + \frac{1}{6}(38n^3 + 15n^2 - 5n)\tau_M + \frac{1}{6}(38n^3 + 3n^2 - 5n)\tau_S\}(N-1)$$
$$+ \{\frac{1}{2}(5n^2 + n)\sigma_D + \frac{1}{6}(38n^3 + 15n^2 - 5n)\sigma_M + \frac{1}{6}(38n^3 + 3n^2 - 5n)\sigma_S\}(m-1)$$
$$+ \{\frac{1}{2}(n^2 + n)t_D + \frac{1}{6}(2n^3 + 3n^2 - 5n)(t_M + t_S)\}$$

(b) with STAR data: $\frac{1}{4}(38n^3 + 31n^2 - n)(N-1) +$
$\frac{1}{6}(8740n^3 + 5103n^2 - 649n)(m-1) +$
$(10n^3 + 38n^2 - 2n)$

(c) for $n = 2$, with STAR data: $106.5N + 14839m - 14717.5$

2. with loop control and storage manipulation (compress, merge, transmit) using STAR data,

(a) for general n, add $80m + \frac{1}{4}(55n^2 + 43n)(N-1) + (182n + 950)(m-1)$

(b) for $n = 2$, add $76.5n + 1394m - 1390.5$, so

total time = $183N + 16233m - 16108$

Notes:

1. An estimate for $\|B[A]\|$ may be computed in the first time through the first loop at a cost of $((2n-1)\tau_+ + \tau_{max})(Nn/2) + ((2n-1)\sigma_+ + \sigma_{max})$. This yields 7.5N + 308 for n = 2 on STAR. The estimate of $\|B[A]\|$ is simply the maximum row sum taken over every other block row. The true value of $\|[A]\|$ could be found by computing the remaining rows of B[A], but this is more expensive and defeats the purpose of odd-even reduction.

4. <u>p-fold reduction</u>

The algorithm and timings are given for $N = p^{\bar{m}} - 1$, $p \geq 3$. For general N, take $\bar{m} = \lceil \log_p N \rceil \doteq m/\log_2 p$, $m = \lceil \log_2 N \rceil$; $\bar{m} = \lceil \log_p N+1 \rceil$ is actually the proper choice.

Algorithm: (see Section 6.A for details), $N_i = p^{\bar{m}+1-i} - 1$.

```
for $i = 1, \ldots, \bar{m}-1$
    for each $k = 1, \ldots, N_{i+1} + 1$,
        LU process on $\Delta_k$
    for each $k = 1, \ldots, N_{i+1}$
        UL process on $\Delta_{k+1}$
    generate $A^{(i+1)}$, $w^{(i+1)}$
solve $b_1^{(\bar{m})} x_1^{(\bar{m})} = w_1^{(\bar{m})}$
for $i = \bar{m}-1, \ldots, 1$
    for each $k = 1, \ldots, N_{i+1} + 1$
        back substitute for $x_{kp-j}$, $1 \leq j \leq p-1$
```

Timing:

1. (a) for general n,
$$\{(5n^2 + n)\tau_D + \tfrac{1}{3}(26n^3 + 3n^2 - 5n)\tau_M + \tfrac{1}{3}(26n^3 - 3n^2 - 5n)\tau_S\}\ (N-p+1)$$
$$+ \{5n^2 + n)\sigma_D + \tfrac{1}{3}(26n^3 + 3n^2 - 5n)\sigma_M + \tfrac{1}{3}(26n^3 - 3n^2 - 5n)\sigma_S\}(\bar{m}-1)(p-1)$$
$$+ \{\tfrac{1}{2}(n^2 + n)t_D + \tfrac{1}{6}(2n^3 + 3n^2 - 5n)(t_M+t_S)\}$$

(b) with STAR data,
$$\tfrac{1}{2}(26n^3 + 21n^2 - n)(N - p + 1) + \tfrac{1}{3}(5980n^3 + 2769n^2 - 649n)\ (\bar{m}-1)(p-1) + (10n^3 + 38n^2 - 2n)$$

(c) using STAR data, $n = 2$: $145N + (19206\bar{m} - 19351)(p-1) + 228$

$$\doteq 145N + 19206m\left(\frac{p-1}{\log_2 p}\right) - (19351p - 19579)$$

2. data manipulation costs are outlined in Section 10.B. We have not derived the complete timing formulae, as the method is expected to be too slow.

Notes:

1. In the arithmetic timing, for $p \geq 3$, p-fold reduction is always cheaper than $(p+1)$-fold reduction for any n. Comparison of 2-fold (odd-even) and 3-fold reduction yields an $\tilde{N}(n)$ such that if $N \geq \tilde{N}(n)$ then odd-even reduction is faster. Although we have not determined $\tilde{N}(n)$, the block LU factorization will probably be faster than either reduction method for $N < \tilde{N}(n)$. Similar results should hold when overhead costs are included in the time estimates.

5. Mixed odd-even reduction and LU factorization

Algorithm: do k odd-even reductions, solve $A^{(k+1)} x^{(k+1)} = w^{(k+1)}$ by the LU factorization, and back substitute.

Timing: (based on $N = 2^m - 1$)

1. (a)
$$\{\tfrac{1}{2}(5n^2 + n)\tau_D + \tfrac{1}{6}(38n^3 + 15n^2 - 5n)\tau_M + \tfrac{1}{6}(38n^3 + 3n^2 - 5n)\tau_S\}(N + 1 - 2^{m-k})$$
$$+ \{\tfrac{1}{2}(5n^2 + n)\sigma_D + \tfrac{1}{6}(38n^3 + 15n^2 - 5n)\sigma_M + \tfrac{1}{6}(38n^3 + 3n^2 - 5n)\sigma_S\}k$$
$$+ \{\tfrac{1}{2}(3n^2 + n)t_D + \tfrac{1}{6}(14n^3 + 9n^2 - 5n)(t_M + t_S)\}(2^{m-k}-1)$$
$$- \{n^2 t_D + (2n^3 + n^2)(t_M + t_S)\}$$

The timing difference between odd-even reduction and the mixed algorithm will be an expression of the form $\alpha_n(m-k) - \beta_n(2^{m-k}) - \gamma_n$, and k should be chosen to maximize this expression. For n = 2 using STAR data, we have $\alpha_n = 14839$, $\beta_n = 905.5$, $\gamma_n = 13028$, $k = m-5$, yielding

(c) $106.5N + 14839m - 46908.5$

2. For n = 2 and STAR data, the polyalgorithm time is $183N + 16233k + 417(2^{m-k}) + 33$, and the optimal choice is $k = m-6$, yielding

(b) $183N + 16233m - 70677$

6. Incomplete reduction

Algorithm: do k odd-even reductions, solve $D[A^{(k+1)}]\ y^{(k+1)} = w^{(k+1)}$, and back substitute to obtain $y = y^{(1)} \approx x^{(1)} = x$.

Timing: (based on $N = 2^m - 1$)

1. (a)

$$\{\tfrac{1}{2}(5n^2 + n)\tau_D + \tfrac{1}{6}(38n^3 + 15n^2 - 5n)\tau_M + \tfrac{1}{6}(38n^3 + 3n^2 - 5n)\tau_S\}\ (N + 1 - 2^{m-k})$$
$$+ \{\tfrac{1}{2}(5n^2 + n)\sigma_D + \tfrac{1}{6}(38n^3 + 15n^2 - 5n)\sigma_M + \tfrac{1}{6}(38n^3 + 3n^2 - 5n)\sigma_S\}k$$
$$+ \{\tfrac{1}{2}(n^2 + n)\tau_D + \tfrac{1}{6}(2n^3 + 3n^2 - 5n)(\tau_M + \tau_S)\}(2^{m-k} - 1)$$
$$+ \{\tfrac{1}{2}(n^2 + n)\sigma_D + \tfrac{1}{6}(2n^3 + 3n^2 - 5n)(\sigma_M + \sigma_S)\}$$

(b) general n, STAR data:

$$\tfrac{1}{4}(38n^3 + 31n^2 - n)N + \tfrac{1}{6}(8740n^3 + 5103n^2 - 649n)\ k - (9n^3 + 6n^2)(2^{m-k} - 1) + \tfrac{1}{6}(460n^3 + 1191n^2 - 649n)$$

(c) using STAR data, $n = 2$: $106.5N + 14839k - 96(2^{m-k}) + 1287$

2. (a) general n, STAR data:

$$\tfrac{1}{4}(38n^3 + 86n^2 + 42n)N + \tfrac{1}{6}(8740n^3 + 5103n^2 + 443n + 5700)k - \tfrac{1}{4}(36n^3 + 79n^2 + 51n)(2^{m-k} - 1) + \tfrac{1}{6}(460n^3 + 1191n^2 - 1651n) + 80(k+1)$$

(b) $n = 2$, STAR data:

$183N + 16233k - 176.5(2^{m-k}) + 1113.5$

7. Jacobi iteration, semi-iteration

basic step: $y^{(i+1)} = B[A]y^{(i)} + D[A]^{-1}v$

for each $j = 1, \ldots, N$

solve $b_j y_j^{(i+1)} = v_j - a_j y_{j-1}^{(i)} - c_j y_{j+1}^{(i)}$

Timing (identical for both vector computer models):

1. (a) $\frac{1}{2}(n^2 + n)(\tau_D N + \sigma_D) + \frac{1}{6}(2n^3 + 15n^2 - 5n)((\tau_M + \tau_S)N + (\sigma_M + \sigma_S))$

1. (b) = 2. (a) using STAR data,

$\frac{1}{4}(2n^3 + 19n^2 - n)N + \frac{1}{6}(460n^3 + 3951n^2 - 649n)$

1. (c) = 2. (b) using STAR data, $n = 2$: $22.5N + 3031$

Optional preprocessing, times for $n = 2$, STAR data

(1) solve $b_j(\alpha_j\ \gamma_j\ \varphi_j) = (a_j\ c_j\ v_j)$,

cost = $38.5N + 4367$. The basic step is then

$y_j^{(i+1)} = \varphi_j - \alpha_j y_{j-1}^{(i)} - \gamma_j y_{j+1}^{(i)}$,

cost = $12N + 1840$.

(2) factor b_j, cost = $3.5N + 367$. The basic step is unchanged from the original, but cost is $19N + 2634$. This is used in the mixed incomplete reduction/Jacobi algorithm.

Semi-iteration, basic step:

$\hat{y}^{(m)}$ = result of Jacobi step from $y^{(m)}$

$y^{(m)} = \omega_{m+1}(\hat{y}^{(m)} - y^{(m-1)}) + y^{(m-1)}$

$\omega_{m+1} = 1/(1 - (\rho^2/4)\omega_m)$,

$\rho = \rho(B[A])$.

Timing, n = 2, using STAR data: 26.5N + 3408, or 16N + 2217 with the first preprocessing.

Notes:

1. Varga [V3, p. 139] shows that for Jacobi semi-iteration,

$$\| e^{(i)} \| \le \left(\frac{2(\omega_b - 1)^{i/2}}{1+(\omega_b - 1)^i} \right) \| e^{(0)} \|,$$

$$\omega_b = 1 + \left(\frac{\rho}{1+\sqrt{1-\rho^2}} \right)^2,$$

$$e^{(i)} = y^{(i)} - x.$$

A conservative estimate for the number of iterations needed to obtain $\| e^{(n)} \| \le 1/N \| e^{(0)} \|$ is $n \doteq 2\log N / (-\log(\omega_b - 1))$. It follows that the execution time for an iterative solution is O(N log N) vs. O(N) for the better direct methods.

8. Incomplete reduction plus Jacobi iterations

Algorithm: do k odd-even reductions, solve $D[A^{(k+1)}]y^{(k+1)} = w^{(k+1)}$, do ℓ Jacobi iterations to improve $y^{(k+1)}$, and back substitute.

As a result of solving $D[A^{(k+1)}]y^{(k+1)} = w^{(k+1)}$, we have factors for $D[A^{(k+1)}]$. This means the cost is (using STAR data, n = 2)

1. (c) $106.5N + 14839k - 96(2^{m-k}) + 1287 + \ell(19(2^{m-k}) + 2615)$.

2. (b) $183N + 16233k - 176.5(2^{m-k}) + 1113.5$

$\qquad + \ell(19(2^{m-k}) + 2615)$.

REFERENCES

[B1] I. Babuska, "Numerical stability in mathematical analysis," IFIP Congress 1968, North-Holland, Amsterdam, pp. 11-23.

[B2] I. Babuska, Numerical stability in the solution of tridiagonal matrices, Rept. BN-609, Inst. for Fluid Dynamics and Appl. Math., Univ. of Md., 1969.

[B3] I. Babuska, "Numerical stability in problems of linear algebra," SIAM J. Numer. Anal., vol. 9, 1972, pp. 53-77.

[B4] D. W. Bailey and D. E. Crabtree, "Bounds for determinants," Lin. Alg. Appl., vol. 2, 1969, pp. 303-309.

[B5] R. E. Bank, "Marching algorithms and block Gaussian elimination," in [B16, pp. 293-307].

[B6] G. H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, R. A. Stoker, "The Illiac IV computer," IEEE Trans. on Comp., vol. C-17, 1968, pp. 746-757.

[B7] G. M. Baudet, Asynchronous iterative methods for multiprocessors, Dept. of Computer Science, Carnegie-Mellon Univ., Nov. 1976.

[B8] F. L. Bauer, "Der Newton-Prozess als quadratisch konvergente Abkurzung des allgemeinen linearen stationaren Iterationsverfahrens 1. Ordnung (Wittmeyer-Prozess)," Z. Angew. Math. Mech., vol. 35, 1955, pp. 469-470.

[B9] A. Benson and D. J. Evans, "The successive peripheral block over-relaxation method," J. Inst. Maths. Applics., vol. 9, 1972, pp. 68-79.

[B10] A. Benson and D. J. Evans, "Successive peripheral overrelaxation and other block methods," J. Comp. Phys., vol. 21, 1976, pp. 1-19.

[B11] G. Birkhoff and A. George, "Elimination by nested dissection," in [T1, pp. 221-269].

[B12] W. J. Bouknight, S. A. Denenberg, D. E. McIntyre, J. M. Randall, A. H. Sameh, D. L. Slotnick, "The Illiac IV system," Proc. IEEE, vol. 60, 1972, pp. 369-388.

[B13] A. Brandt, Multi-level adaptive techniques I. The multi-grid method, Rept. RC 6026, IBM T. J. Watson Res. Cen., Yorktown Heights, N. Y., 1976.

[B14] J. L. Brenner, "A bound for a determinant with dominant main diagonal," Proc. AMS, vol. 5, 1954, pp. 631-634.

[B15] C. G. Broyden, "Some condition number bounds for the Gaussian elimination process," J. Inst. Maths. Applics., vol. 12, 1973, pp. 273-286.

[B16] J. R. Bunch and D. J. Rose, eds., Sparse Matrix Computations, Academic Press, N. Y., 1976.

[B17] B. L. Buzbee, Application of fast Poisson solvers to the numerical approximation of parabolic problems, Rept. LA-4950-T, Los Alamos Sci. Lab., 1972.

[B18] B. L. Buzbee and A. Carasso, "On the numerical computation of parabolic problems for preceding times," Math. Comp., vol. 27, 1973, pp. 237-266.

[B19] B. L. Buzbee, F. W. Dorr, J. A. George and G. H. Golub, "The direct solution of the discrete Poisson equation on irregular regions," SIAM J. Numer. Anal., vol. 8, 1971, pp. 722-736.

[B20] B. L. Buzbee, G. H. Golub and C. W. Nielson, "On direct methods for solving Poisson's equations," SIAM J. Numer. Anal., vol. 7, 1970, pp. 627-656.

[C1] A. Carasso, "The abstract backward beam equation," SIAM J. Math. Anal., vol. 2, 1971, pp. 193-212.

[C2] Control Data Corporation, CDC STAR-100 Instruction execution times, Arden Hills, Minn., Jan. 1976; in conjunction with measurements made at Lawrence Livermore Laboratory.

[C3] A. K. Cline, personal communication, 1974.

[C4] P. Concus and G. H. Golub, "Use of fast direct methods for the efficient solution of nonseparable elliptic equations," SIAM J. Numer. Anal., vol. 10, 1973, pp. 1103-1120.

[C5] P. Concus, G. H. Golub and D. P. O'Leary, "A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations," in [B16, pp. 309-332].

[C6] S. D. Conte and C. de Boor, Elementary Numerical Analysis, McGraw-Hill, N. Y., 1972.

[C7] M. G. Cox, "Curve fitting with piecewise polynomials," J. Inst. Maths. Applics., vol. 8, 1971, pp. 36-52.

[C8] Cray Research, Inc., "Cray-1 Computer," Chippewa Falls, Wis., 1975.

[D1] M. A. Diamond and D. L. V. Ferreira, "On a cyclic reduction method for the solution of Poisson's equations," SIAM J. Numer. Anal., vol. 13, 1976, pp. 54-70.

[D2] F. W. Dorr, "The direct solution of the discrete Poisson equation on a rectangle," SIAM Review, vol. 12, 1970, pp. 248-263.

[D3] F. W. Dorr, "An example of ill-conditioning in the numerical solution of singular perturbation problems," Math. Comp., vol. 25, 1971, pp. 271-283.

[E1] S. C. Eisenstat, M. H. Schultz and A. H. Sherman, "Applications of an element model for Gaussian elimination," in [B16, pp. 85-96].

[E2] D. J. Evans, "A new iterative method for the solution of sparse systems of linear difference equations," in [R6, pp. 89-100].

[E3] R. K. Even and Y. Wallach, "On the direct solution of Dirichlet's problem in two dimensions," Computing, vol. 5, 1970, pp. 45-56.

[F1] D. C. Feingold and R. S. Varga, "Block diagonally dominant matrices and generalizations of the Gerschgorin circle theorem," Pacific J. Math., vol. 12, 1962, pp. 1241-1250.

[F2] D. Fischer, G. Golub, O. Hald, C. Leiva and O. Widlund, "On Fourier-Toeplitz methods for separable elliptic problems," Math. Comp., vol. 28, 1974, pp. 349-368.

[F3] G. von Fuchs, J. R. Roy and E. Schrem, "Hypermatrix solution of large sets of symmetric positive definite linear equations," Comp. Math. in Appl. Mech. and Engng., vol. 1, 1972, pp. 197-216.

[G1] J. A. George, "Nested dissection of a regular finite element mesh," SIAM J. Numer. Anal., vol. 10, 1973, pp. 345-363.

[H1] L. A. Hageman and R. S. Varga, "Block iterative methods for cyclically reduced matrix equations," Numer. Math., vol. 6, 1964, pp. 106-119.

[H2] D. E. Heller, "Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems," SIAM J. Numer. Anal., vol. 13, 1976, pp. 484-496.

[H3] D. E. Heller, A survey of parallel algorithms in numerical linear algebra, Dept. of Comp. Sci., Carnegie-Mellon Univ., 1976; SIAM Review, to appear.

[H4] D. E. Heller, D. K. Stevenson and J. F. Traub, "Accelerated iterative methods for the solution of tridiagonal systems on parallel computers," J.ACM, vol. 23, 1976, pp. 636-654.

[H5] R. W. Hockney, "A fast direct solution of Poisson's equation using Fourier analysis," J.ACM, vol. 12, 1965, pp. 95-113.

[H6] R. W. Hockney, "The potential calculation and some applications," Methods in Computational Physics, vol. 9, 1970, pp. 135-211.

[H7] E. Horowitz, "The application of symbolic mathematics to a singular perturbation problem," Proc. ACM Annual Conf., 1972, vol. II, pp. 816-825.

[H8] A. S. Householder, The Theory of Matrices in Numerical Analysis, Blaisdell, N. Y., 1964.

[J1] T. L. Jordan, A new parallel algorithm for diagonally dominant tridiagonal matrices, Los Alamos Sci. Lab., 1974.

[J2] M. L. Juncosa and T. W. Mullikin, "On the increase of convergence rates of relaxation procedures for elliptic partial difference equations," J.ACM, vol. 7, 1960, pp. 29-36.

[K1] W. Kahan, Conserving confluence curbs ill-condition, Dept. of Comp. Sci., Univ. of Calif., Berkeley, 1972.

[K2] H. T. Kung, "On computing reciprocals of power series, " Numer. Math., vol. 22, 1974, pp. 341-348.

[K3] H. T. Kung and J. F. Traub, All algebraic functions can be computed fast, Dept. of Comp. Sci., Carnegie-Mellon Univ., July 1976.

[L1] J. J. Lambiotte, Jr., The solution of linear systems of equations on a vector computer, Dissertation, Univ. of Virginia, 1975.

[L2] J. J. Lambiotte, Jr. and R. G. Voigt, "The solution of tridiagonal linear systems on the CDC STAR-100 computer," ACM Trans. Math. Software, vol. 1, 1975, pp. 308-329.

[M1] N. Madsen and G. Rodrigue, A comparison of direct methods for tri-diagonal systems on the CDC STAR-100, Lawrence Livermore Lab., 1975.

[M2] M. A. Malcolm and J. Palmer, "A fast direct method for solving a class of tridiagonal linear systems," CACM, vol. 17, 1974, pp. 14-17.

[M3] W. Miller, "Software for roundoff analysis," ACM Trans. Math. Software, vol. 1, 1975, pp. 108-128.

[N1] A. K. Noor and S. J. Voigt, "Hypermatrix scheme for finite element systems on CDC STAR-100 computer," Computers and Structures, vol. 5, 1975, pp. 287-296.

[O1] J. M. Ortega and R. G. Voigt, Solution of partial differential equations on vector computers, ICASE, Hamton, Va., 1977.

[O2] A. M. Ostrowski, "Über die Determinanten mit überwiegender Haupt-diagonale," Comment. Math. Helv., vol. 10, 1937, pp. 69-96.

[P1] G. Peters and J. H. Wilkinson, "On the stability of Gauss-Jordan elimination with pivoting," CACM, vol. 18, 1975, pp. 20-24.

[R1] J. K. Reid, ed., Large Sparse Sets of Linear Equations, Academic Press, London, 1970.

[R2] J. K. Reid, "On the method of conjugate gradients for the solution of large sparse systems of linear equations," in [R1, pp. 231-254].

[R3] F. Robert, "Blocks-H-matrices et convergence des methods iteratives classiques par blocks," Lin. Alg. Appl., vol. 2, 1960, pp. 223-265.

[R4] D. J. Rose and J. R. Bunch, "The role of partitioning in the numerical solution of sparse systems," in [R6, pp. 177-187].

[R5] D. J. Rose and G. F. Whitten, "A recursive analysis of dissection strategies," in [B16, pp. 59-83].

[R6] D. J. Rose and R. A. Willoughby, eds., Sparse Matrices and Their Applications, Plenum Press, N. Y., 1972.

[R7] J. B. Rosser, The direct solution of difference analogs of Poisson's equation, rep. MRC-TSR-797, Math. Res. Cen., Madison, Wis., 1967.

[S1] J. Schröder and U. Trottenberg, "Reduktionsverfahren fur Differenzengleichungen bei Randwertaufgaben I," Numer. Math., vol. 22, 1973, pp. 37-68.

[S1a] J. Schröder, U. Trottenberg and H. Reutersberg, "Reduktionsverfahren für Differenzengleichungen bei Randwertaufgaben II," Numer. Math., vol. 16, 1976, pp. 429-459.

[S2] H. S. Stone, "Parallel tridiagonal equation solvers," ACM Trans. Math. Software, vol. 1, 1975, pp. 289-307.

[S3] H. S. Stone, ed., Introduction to Computer Architecture, Science Research Associates, Palo Alto, Calif., 1975.

[S4] G. Strang and G. J. Fix, An Analysis of the Finite Element Method, Prentice-Hall, Englewood Cliffs, N. J., 1973.

[S5] P. N. Swarztrauber, "The direct solution of the discrete Poisson equation on the surface of a sphere," J. Comp. Phys., vol. 15, 1974, pp. 46-54.

[S6] P. N. Swarztrauber, "A direct method for the discrete solution of separable elliptic equations," SIAM J. Numer. Anal., vol. 11, 1974, pp. 1136-1150.

[S7] P. N. Swarztrauber and R. A. Sweet, "The direct solution of the discrete Poisson equation on a disc," SIAM J. Numer. Anal., vol. 10, 1973, pp. 900-907.

[S8] P. Swarztrauber and R. Sweet, Efficient FORTRAN subprograms for the solution of elliptic partial differential equations, Rept. NCAR-TN/IA-109, Nat. Cen. for Atmospheric Res., Boulder, Col., 1975.

[S9] R. A. Sweet, "Direct methods for the solution of Poisson's equation on a staggered grid," J. Comp. Phys., vol. 12, 1973, pp. 422-428.

[S10] R. A. Sweet, "A generalized cyclic reduction algorithm," SIAM J. Numer. Anal., vol. 11, 1974, pp. 506-520.

[S11] R. A. Sweet, "A cyclic reduction algorithm for block tridiagonal systems of arbitrary dimension," contributed paper, Symp. on Sparse Matrix Computations, Argonne Nat. Lab., Sept. 1975.

[T1] J. F. Traub, ed., Complexity of Sequential and Parallel Numerical Algorithms, Academic Press, N. Y., 1973.

[T2] J. F. Traub, "Iterative solution of tridiagonal systems on parallel or vector computers," in [T1, pp. 49-82].

[T3] J. F. Traub, ed., Analytic Computational Complexity, Academic Press, N. Y., 1976.

[V1] J. M. Varah, "On the solution of block tridiagonal systems arising from certain finite difference equations," Math. Comp., vol. 26, 1972, pp. 859-868.

[V2] J. M. Varah, "A comparison of some numerical methods for two-point boundary value problems," Math. Comp., vol. 28, 1974, pp. 743-755.

[V3] R. S. Varga, Matrix Iterative Analysis, Prentice-Hall, Englewood Cliffs, N. J., 1962.

[V4] R. S. Varga, "On recurring theorems on diagonal dominance," Lin. Alg. Appl., vol. 13, 1976, pp. 1-9.

[W1] W. J. Watson, "The TI ASC, a highly modular and flexible supercomputer architecture," AFIPS Fall 1972, AFIPS Press, Montvale, N. J., vol. 41, pt. 1, pp. 221-229.

[W2] O. B. Widlund, "On the use of fast methods for separable finite difference equations for the solution of general elliptic problems," in [R6, pp. 121-131].

[W3] J. H. Wilkinson, "Error analysis of direct methods of matrix inversion," J.ACM, vol. 8, 1961, pp. 281-330.

[W4] W. A. Wulf and C. G. Bell, "C.mmp, a multi-mini-processor," AFIPS Fall 1972, AFIPS Press, Montvale, N. J., vol. 41, pt. 2, pp. 765-777.

[y1] D. Y. Y. Yun, "Hensel meets Newton - algebraic constructions in an analytic setting," in [T3, pp. 205-215].

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DIRECT AND ITERATIVE METHODS FOR BLOCK TRIDIAGONAL LINEAR SYSTEMS		5. TYPE OF REPORT & PERIOD COVERED Interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Don Eric Heller		8. CONTRACT OR GRANT NUMBER(s) MCS75-222-55 N00014-76-C-0370; NR 044-422
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Computer Science Dept. Pittsburgh, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Arlington, VA 22217		12. REPORT DATE April 1977
		13. NUMBER OF PAGES 169
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Block tridiagonal systems of linear equations occur frequently in scientific computations, often forming the core of more complicated problems. Numerical methods for solution of such systems are studied with emphasis on efficient methods for a vector computer. A convergence theory for direct methods under conditions of block diagonal dominace is developed, demonstrating stability, convergence and approximation properties of direct methods. Block elimination (LU factorization) is linear, cyclic odd-even reduction is quadratic, and higher-order methods exist. The odd-even methods are variations of the quadratic Newton iteration for the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

continued

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. abstract continued
inverse matrix, and are the only quadratic methods within a certain reasonable class of algorithms. Semi-direct methods based on the quadratic convergence of odd-even reduction prove useful in combination with linear iterations for an approximate solution. An execution time analysis for a pipeline computer is given, with attention to storage requirements and the effect of machine constraints on vector operations.